Course: Deep Learning

Name: SAYEDPEDRAM HAERI BOROUJENI

Instructor: Dr. Feng Luo

Home Work: Number One

Part 1: Deep vs Shallow

1. Import My Packages

In [6]:
```
import os
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.backends.cudnn as cudnn
import torchvision.transforms as transformtransforms
from torchvision import models
from torchsummary import summary
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import ToPILImage
from tqdm import tqdm
import copy
import math
import matplotlib.pyplot as plt
import numpy as np
import torchvision.transforms.functional as TF
from torchvision import transforms
import cv2
import random
from PIL import Image
from glob import glob
from torch import nn

os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
TORCH_CUDA_ARCH_LIST="8.6"

Project_PATH = os.path.dirname(os.path.abspath('__file__'))
outputs_dir = Project_PATH + '/Desktop/Deep Learning HW1'
model_path = Project_PATH + '/My Model/'
```

2. My Device

In [7]:
```
device_default = torch.cuda.current_device()
torch.cuda.device(device_default)
device = torch.device("cuda")
print("torch.cuda.is_available:", torch.cuda.is_available())
```

```
print("torch.cuda.device_count:", torch.cuda.device_count())
print("torch.cuda.current_device:", torch.cuda.current_device())
print("torch.cuda.get_device_name:", torch.cuda.get_device_name(device_default))
print("torch.version.cuda:", torch.version.cuda)
print("torch.version:", torch.__version__)
print("torch.cuda.arch_list:", torch.cuda.get_arch_list())
```

```
torch.cuda.is_available: True
torch.cuda.device_count: 1
torch.cuda.current_device: 0
torch.cuda.get_device_name: NVIDIA RTX A5000
torch.version.cuda: 11.3
torch.version: 1.11.0
torch.cuda.arch_list: ['sm_37', 'sm_50', 'sm_60', 'sm_61', 'sm_70', 'sm_75', 'sm_80',
'sm_86', 'compute_37']
```

3. My Models: DNN with 8 Layers, 5 Layers, 2 Layers

In [8]:

```
# DNN with 8 Layers
class DNN_8L_Model(nn.Module):
    def __init__(self):
        super(DNN_8L_Model, self, ).__init__()
        self.layer1 = nn.Sequential(nn.Linear(1, 5),nn.ReLU(True))
        self.layer2 = nn.Sequential(nn.Linear(5, 10),nn.ReLU(True))
        self.layer3 = nn.Sequential(nn.Linear(10, 10),nn.ReLU(True))
        self.layer4 = nn.Sequential(nn.Linear(10, 10),nn.ReLU(True))
        self.layer5 = nn.Sequential(nn.Linear(10, 10),nn.ReLU(True))
        self.layer6 = nn.Sequential(nn.Linear(10, 10),nn.ReLU(True))
        self.layer7 = nn.Sequential(nn.Linear(10, 5),nn.ReLU(True))
        self.layer8 = nn.Sequential(nn.Linear(5, 1))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        x = self.layer6(x)
        x = self.layer7(x)
        x = self.layer8(x)
        return x

device = torch.device("cuda")
Model_DNN_8 = DNN_8L_Model().to(device)
summary(Model_DNN_8, (1,1))

# DNN with 5 Layers
class DNN_5L_Model(nn.Module):
    def __init__(self):
        super(DNN_5L_Model, self).__init__()
        self.layer1 = nn.Sequential(nn.Linear(1, 10),nn.ReLU(True))
        self.layer2 = nn.Sequential(nn.Linear(10, 18),nn.ReLU(True))
        self.layer3 = nn.Sequential(nn.Linear(18, 15),nn.ReLU(True))
        self.layer4 = nn.Sequential(nn.Linear(15, 4),nn.ReLU(True))
        self.layer5 = nn.Sequential(nn.Linear(4, 1))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
```

```
        x = self.layer5(x)
        return x

device = torch.device("cuda")
Model_DNN_5 = DNN_5L_Model().to(device)
summary(Model_DNN_5, (1,1))

# DNN with 2 Layers
class DNN_2L_Model(nn.Module):
    def __init__(self):
        super(DNN_2L_Model, self).__init__()
        self.layer1 = nn.Sequential(nn.Linear(1, 190),nn.ReLU(True))
        self.layer2 = nn.Sequential(nn.Linear(190, 1))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        return x

device = torch.device("cuda")
Model_DNN_2 = DNN_2L_Model().to(device)
summary(Model_DNN_2, (1,1))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 1, 5]              10
              ReLU-2                 [-1, 1, 5]               0
            Linear-3                [-1, 1, 10]              60
              ReLU-4                [-1, 1, 10]               0
            Linear-5                [-1, 1, 10]             110
              ReLU-6                [-1, 1, 10]               0
            Linear-7                [-1, 1, 10]             110
              ReLU-8                [-1, 1, 10]               0
            Linear-9                [-1, 1, 10]             110
             ReLU-10                [-1, 1, 10]               0
           Linear-11                [-1, 1, 10]             110
             ReLU-12                [-1, 1, 10]               0
           Linear-13                 [-1, 1, 5]              55
             ReLU-14                 [-1, 1, 5]               0
           Linear-15                 [-1, 1, 1]               6
================================================================
Total params: 571
Trainable params: 571
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 10]              20
              ReLU-2                [-1, 1, 10]               0
            Linear-3                [-1, 1, 18]             198
              ReLU-4                [-1, 1, 18]               0
            Linear-5                [-1, 1, 15]             285
              ReLU-6                [-1, 1, 15]               0
            Linear-7                 [-1, 1, 4]              64
              ReLU-8                 [-1, 1, 4]               0
            Linear-9                 [-1, 1, 1]               5
================================================================
Total params: 572
Trainable params: 572
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 190]             380
              ReLU-2               [-1, 1, 190]               0
            Linear-3                 [-1, 1, 1]             191
================================================================
Total params: 571
Trainable params: 571
Non-trainable params: 0
```

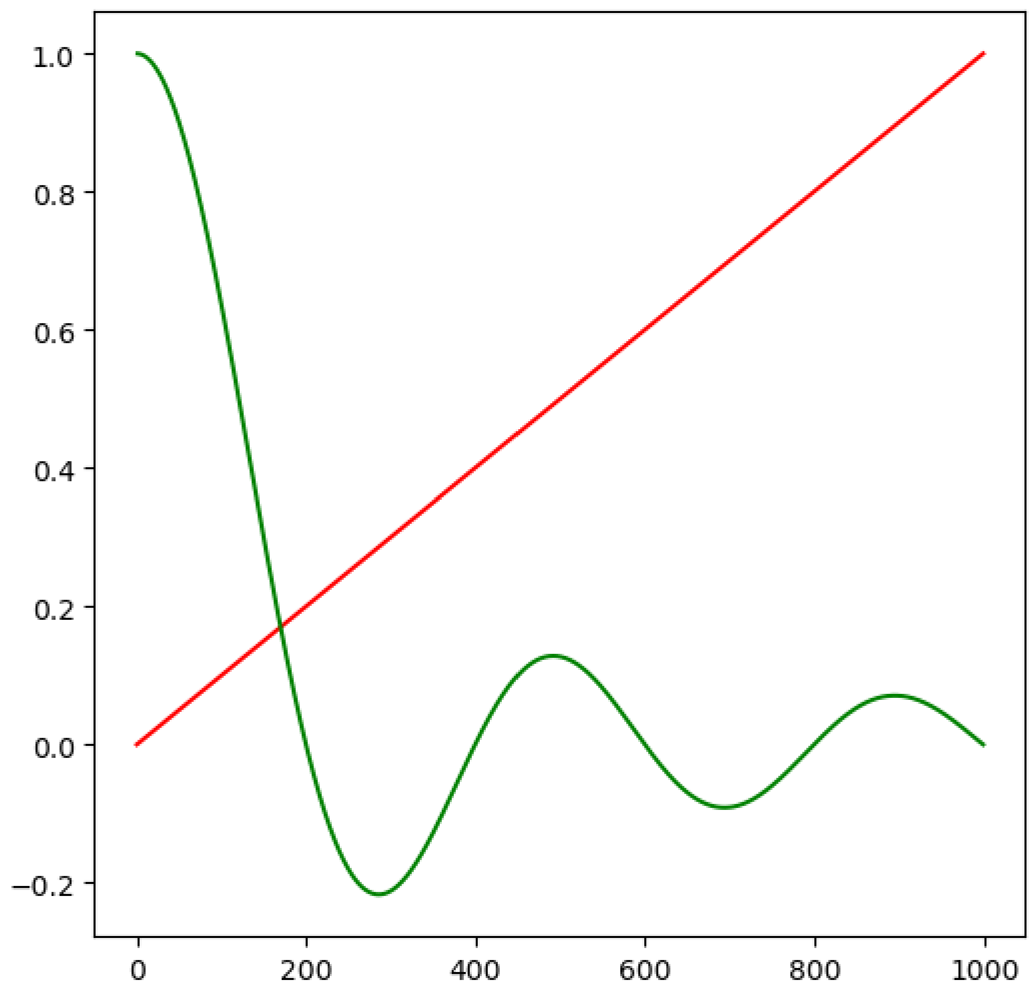
```
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

4. My Function Simulation

4.1. F1(x) = sin(5*np.pi*x)/(5*np.pi*x)

In [10]:
```
x = torch.linspace(0,1,1000).unsqueeze(1)
y = torch.sin(5*np.pi*x)/(5*np.pi*x)
plt.figure(figsize=(6,6))
plt.plot(x,"red")
plt.plot(y, "green")
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x1d534d8fe50>]
```

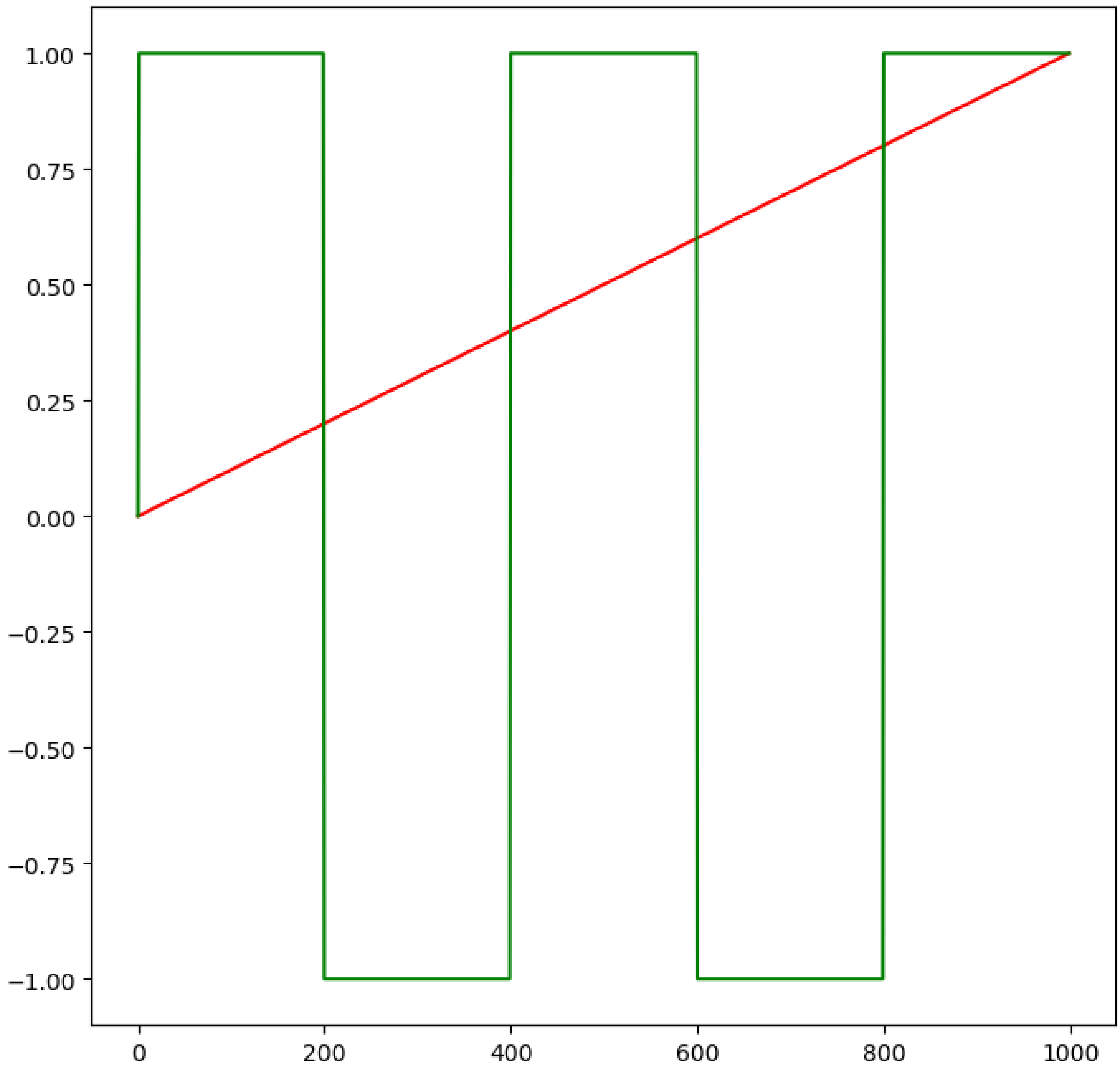


4.2. F2(x) = sgn(torch.sin(5*np.pi*x), 0)

In [11]:
```
def sgn(x, y):
    h,w = list(x.size())
```

```
    Y = torch.rand(h,w)
    for i in range(h):
        if(x[i] > y): Y[i] = 1
        if(x[i] < y): Y[i] = -1
        if(x[i] == y): Y[i] = 0
    return Y

x = torch.linspace(0,1,1000).unsqueeze(1)
y = sgn(torch.sin(5*np.pi*x), 0)
plt.figure(figsize=(8,8))
plt.plot(x, "red")
plt.plot(y, "green")
```

Out[11]: [<matplotlib.lines.Line2D at 0x1d5394bd850>]



4.3. Initialization

In [12]:
```
x = torch.linspace(0,1,1000).unsqueeze(1)
y = torch.sin(5*np.pi*x)/(5*np.pi*x)
y[0] = y[1]
function1 = y
```

```
y = sgn(torch.sin(5*np.pi*x), 0)
function2 = y
```

4.4. Training

In [13]:
```
def train(function,
          model_name,
          Epochs = 20000,
          Batch  = 1000,
          Data_workers = 0,
          LR = 0.0005):

    torch.cuda.is_available()
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    Model = model_name().to(device)
    x = torch.linspace(0,1,1000).unsqueeze(1)
    x = x.to(device)
    y = function.to(device)

    criterion = nn.MSELoss()
    optimizer = optim.Adam(Model.parameters(), lr=LR)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size = 100, gamma = 0.8)

    trainloss_list = []
    lr_list = []

    for epoch in range(Epochs):
        Model.train()
        train_loss = 0.0
        y_pred = Model(x)
        loss = criterion(y_pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss = loss.item()
        trainloss_list.append(train_loss)
        lr_list.append(optimizer.state_dict()['param_groups'][0]['lr'])
        if epoch >= Epochs//2:
            scheduler.step()

        if epoch % (Epochs//10) == 0:
            print('{}/{}, loss: {}'.format(epoch,Epochs,train_loss))

    return [Model,trainloss_list,lr_list]
```

In [14]:
```
[Model_F1_8L,trainloss_F1_8L,lr_F1_8L] = train(function1, DNN_8L_Model, Epochs=5000, E
[Model_F1_5L,trainloss_F1_5L,lr_F1_5L] = train(function1, DNN_5L_Model, Epochs=5000, E
[Model_F1_2L,trainloss_F1_2L,lr_F1_2L] = train(function1, DNN_2L_Model, Epochs=5000, E
```

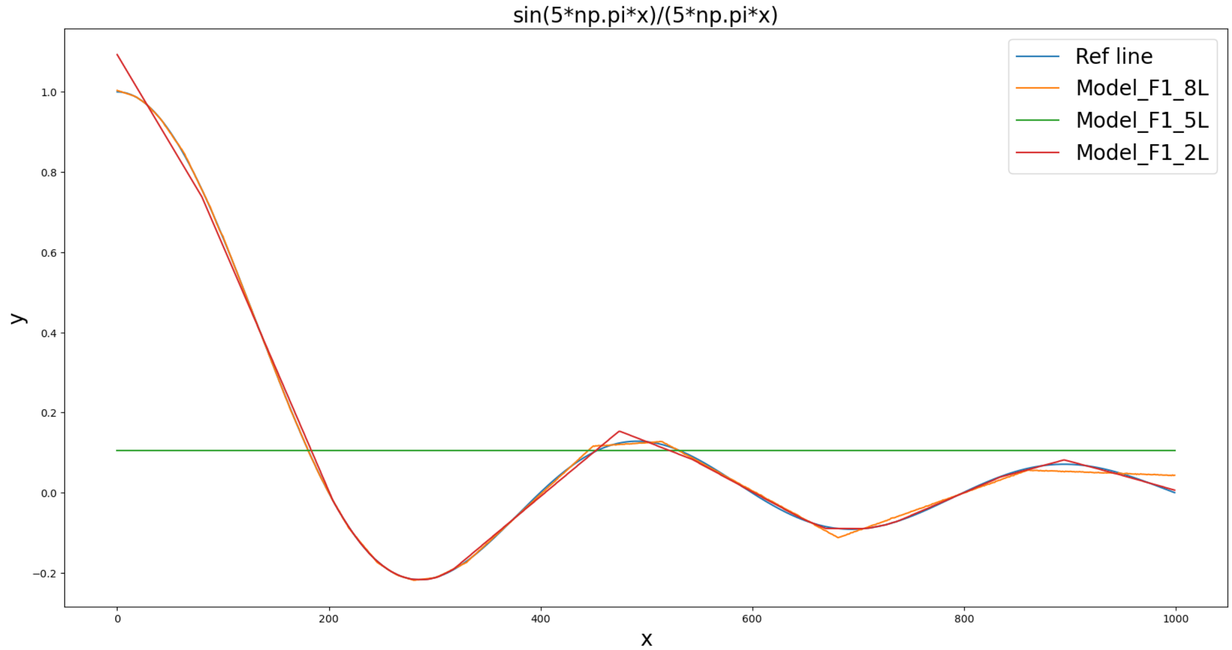
```
0/5000, loss: 0.08884409070014954
500/5000, loss: 0.00031075175502337515
1000/5000, loss: 0.0003412480291444808
1500/5000, loss: 0.00021367848967202008
2000/5000, loss: 0.00016854458954185247
2500/5000, loss: 0.00012329664605204016
3000/5000, loss: 8.833206811686978e-05
3500/5000, loss: 7.859340985305607e-05
4000/5000, loss: 7.503526285290718e-05
4500/5000, loss: 7.292279769899324e-05
0/5000, loss: 0.20190690457820892
500/5000, loss: 0.08747661113739014
1000/5000, loss: 0.08747661113739014
1500/5000, loss: 0.08747661113739014
2000/5000, loss: 0.08747661113739014
2500/5000, loss: 0.08747661113739014
3000/5000, loss: 0.08747661113739014
3500/5000, loss: 0.08747661113739014
4000/5000, loss: 0.08747661113739014
4500/5000, loss: 0.08747661113739014
0/5000, loss: 0.09659905731678009
500/5000, loss: 0.000659201992675662
1000/5000, loss: 0.0004328907234594226
1500/5000, loss: 0.0003688787401188165
2000/5000, loss: 0.0003107521333731711
2500/5000, loss: 0.0002256224543089047
3000/5000, loss: 0.00018621599883772433
3500/5000, loss: 0.0001710306532913819
4000/5000, loss: 0.00016506131214555353
4500/5000, loss: 0.00016257581592071801
```

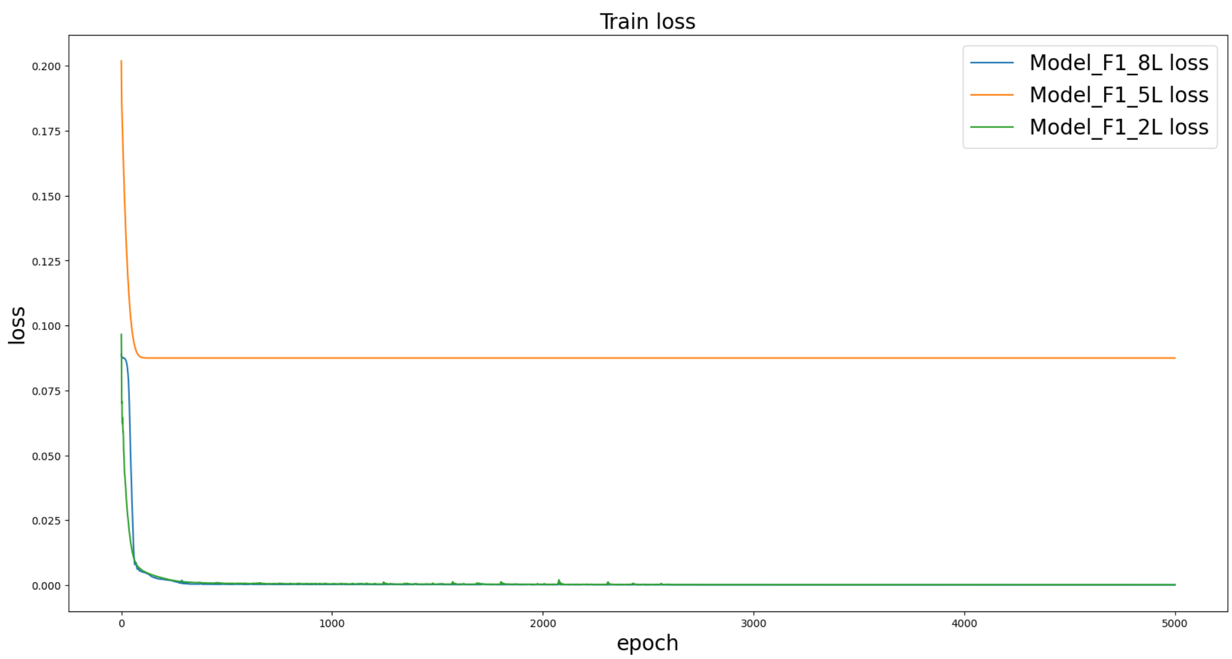
In [15]:

```
torch.cuda.is_available()
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
x = torch.linspace(0,1,1000).unsqueeze(1)
x = x.to(device)
y = function1.to(device)
criterion = nn.MSELoss()

Model_F1_8L.eval()
Model_F1_5L.eval()
Model_F1_2L.eval()
with torch.no_grad():
    y_pred_F1_8L = Model_F1_8L(x)
    y_pred_F1_5L = Model_F1_5L(x)
    y_pred_F1_2L = Model_F1_2L(x)
    testloss_F1_8L = criterion(y_pred_F1_8L, y)
    testloss_F1_5L = criterion(y_pred_F1_5L, y)
    testloss_F1_2L = criterion(y_pred_F1_2L, y)
    print('Model_F1_8L loss: {}'.format(testloss_F1_8L))
    print('Model_F1_5L loss: {}'.format(testloss_F1_5L))
    print('Model_F1_2L loss: {}'.format(testloss_F1_2L))

plt.figure(figsize=(20,10))
plt.plot(y.cpu().numpy(),label='Ref line')
plt.plot(y_pred_F1_8L.detach().cpu().numpy(),label='Model_F1_8L')
plt.plot(y_pred_F1_5L.detach().cpu().numpy(),label='Model_F1_5L')
plt.plot(y_pred_F1_2L.detach().cpu().numpy(),label='Model_F1_2L')
plt.xlabel('x',fontsize=20)
plt.ylabel('y',fontsize=20)
plt.title('sin(5*np.pi*x)/(5*np.pi*x)',fontsize=20)
```

```
plt.legend(fontsize=20)
plt.show()

plt.figure(figsize=(20,10))
plt.plot(trainloss_F1_8L, label='Model_F1_8L loss')
plt.plot(trainloss_F1_5L, label='Model_F1_5L loss')
plt.plot(trainloss_F1_2L, label='Model_F1_2L loss')
plt.xlabel('epoch',fontsize=20)
plt.ylabel('loss',fontsize=20)
plt.title('Train loss',fontsize=20)
plt.legend(fontsize=20)
plt.show()
```

```
Model_F1_8L loss: 7.200584514066577e-05
Model_F1_5L loss: 0.08747661113739014
Model_F1_2L loss: 0.00016157510981429368
```





In [16]:
```
[Model_F2_8L,trainloss_F2_8L,lr_F2_8L] = train(function2, DNN_8L_Model, Epochs=5000, E
[Model_F2_5L,trainloss_F2_5L,lr_F2_5L] = train(function2, DNN_5L_Model, Epochs=5000, E
```

```
[Model_F2_2L,trainloss_F2_2L,lr_F2_2L] = train(function2, DNN_2L_Model, Epochs=5000, E
```

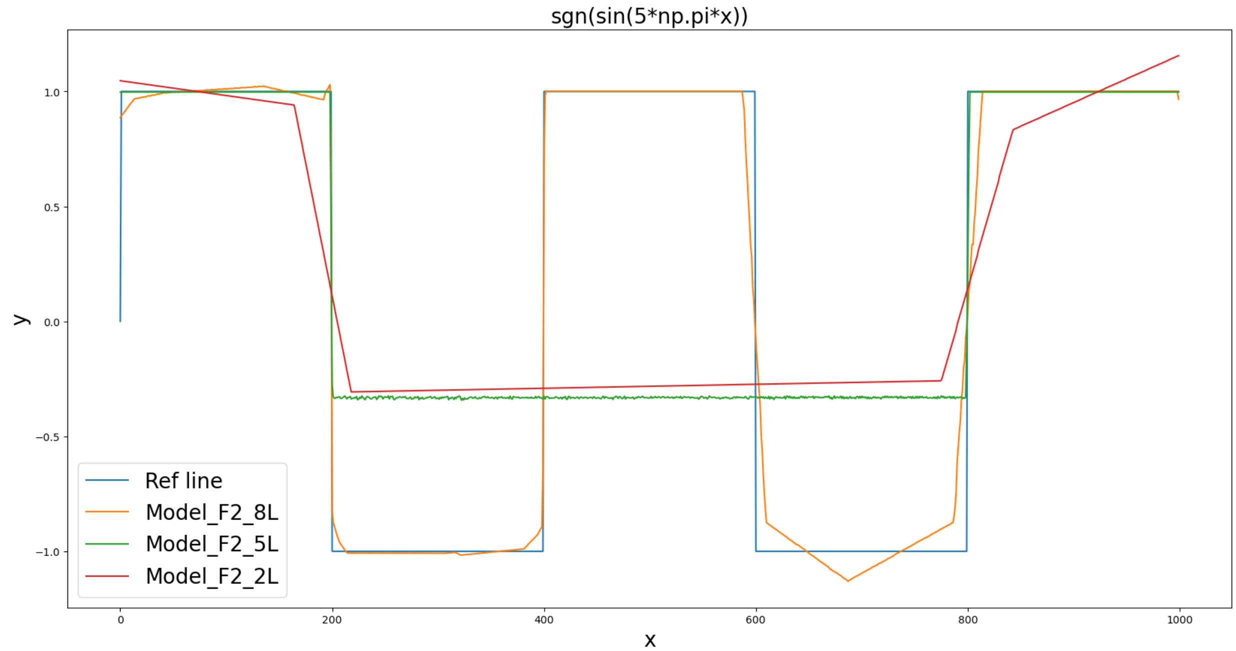
```
0/5000, loss: 0.9600096940994263
500/5000, loss: 0.5428025722503662
1000/5000, loss: 0.5386829376220703
1500/5000, loss: 0.5465179085731506
2000/5000, loss: 0.5596826672554016
2500/5000, loss: 0.06379742175340652
3000/5000, loss: 0.027715761214494705
3500/5000, loss: 0.02180306427180767
4000/5000, loss: 0.020665109157562256
4500/5000, loss: 0.020045064389705658
0/5000, loss: 0.959647536277771
500/5000, loss: 0.5457130670547485
1000/5000, loss: 0.5415025949478149
1500/5000, loss: 0.5382223129272461
2000/5000, loss: 0.5377762317657471
2500/5000, loss: 0.5363697409629822
3000/5000, loss: 0.5359899401664734
3500/5000, loss: 0.5357400178909302
4000/5000, loss: 0.5356199145317078
4500/5000, loss: 0.5357403755187988
0/5000, loss: 1.057806372642517
500/5000, loss: 0.648601233959198
1000/5000, loss: 0.62505042552948
1500/5000, loss: 0.5991261601448059
2000/5000, loss: 0.5882933139801025
2500/5000, loss: 0.581540584564209
3000/5000, loss: 0.5776863098144531
3500/5000, loss: 0.5764163136482239
4000/5000, loss: 0.575908362865448
4500/5000, loss: 0.5757037401199341
```

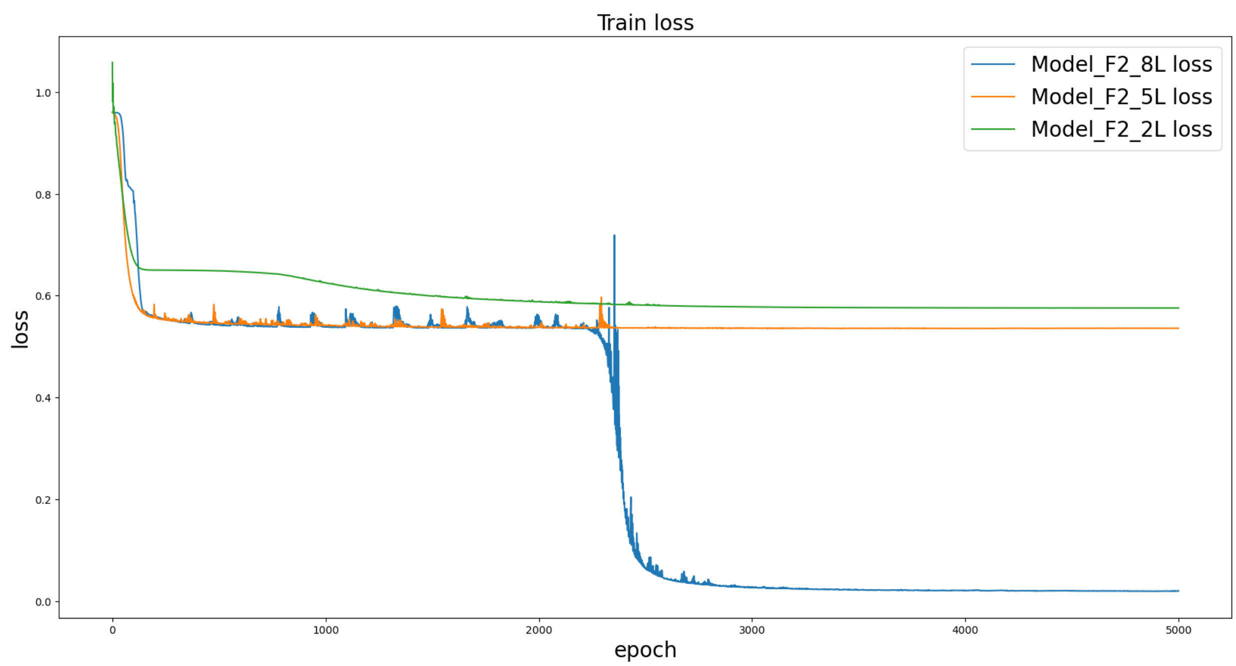
In [17]:

```
torch.cuda.is_available()
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
x = torch.linspace(0,1,1000).unsqueeze(1)
x = x.to(device)
y = function2.to(device)
criterion = nn.MSELoss()

Model_F2_8L.eval()
Model_F2_5L.eval()
Model_F2_2L.eval()
with torch.no_grad():
    y_pred_F2_8L = Model_F2_8L(x)
    y_pred_F2_5L = Model_F2_5L(x)
    y_pred_F2_2L = Model_F2_2L(x)
    testloss_F2_8L = criterion(y_pred_F2_8L, y)
    testloss_F2_5L = criterion(y_pred_F2_5L, y)
    testloss_F2_2L = criterion(y_pred_F2_2L, y)
    print('Model_F2_8L loss: {}'.format(testloss_F2_8L))
    print('Model_F2_5L loss: {}'.format(testloss_F2_5L))
    print('Model_F2_2L loss: {}'.format(testloss_F2_2L))

plt.figure(figsize=(20,10))
plt.plot(y.cpu().numpy(),label='Ref line')
plt.plot(y_pred_F2_8L.detach().cpu().numpy(),label='Model_F2_8L')
plt.plot(y_pred_F2_5L.detach().cpu().numpy(),label='Model_F2_5L')
plt.plot(y_pred_F2_2L.detach().cpu().numpy(),label='Model_F2_2L')
plt.xlabel('x',fontsize=20)
```

```
plt.ylabel('y',fontsize=20)
plt.title('sgn(sin(5*np.pi*x))',fontsize=20)
plt.legend(fontsize=20)
plt.show()

plt.figure(figsize=(20,10))
plt.plot(trainloss_F2_8L, label='Model_F2_8L loss')
plt.plot(trainloss_F2_5L, label='Model_F2_5L loss')
plt.plot(trainloss_F2_2L, label='Model_F2_2L loss')
plt.xlabel('epoch',fontsize=20)
plt.ylabel('loss',fontsize=20)
plt.title('Train loss',fontsize=20)
plt.legend(fontsize=20)
plt.show()
```

```
Model_F2_8L loss: 0.02047920413315296
Model_F2_5L loss: 0.535820722579956
Model_F2_2L loss: 0.5756163597106934
```





5. My Models: CNN with 6 Layers, 5 Layers, 4 Layers

Dataset: CIFAR

In [18]:
```
# CNN with 6 Layers
class CNN_6L_Model(nn.Module):
    def __init__(self):
        super(CNN_6L_Model, self).__init__()
        self.layer1 = nn.Sequential(nn.Conv2d(3, 10, 3),nn.ReLU(True),nn.MaxPool2d(ker
        self.layer2 = nn.Sequential(nn.Conv2d(10, 16, 3),nn.ReLU(True),nn.MaxPool2d(ke
        self.layer3 = nn.Sequential(nn.Conv2d(16, 32, 3),nn.ReLU(True))
        self.layer4 = nn.Sequential(nn.Linear(32*4*4, 128),nn.ReLU(True))
        self.layer5 = nn.Sequential(nn.Linear(128, 32),nn.ReLU(True))
        self.layer6 = nn.Sequential(nn.Linear(32, 10))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = x.view(x.size()[0], -1)
        x = self.layer4(x)
        x = self.layer5(x)
        x = self.layer6(x)
        return x

device = torch.device("cuda")
Model = CNN_6L_Model().to(device)
summary(Model, input_size=(3,32,32))

# CNN with 5 Layers
class CNN_5L_Model(nn.Module):
    def __init__(self):
        super(CNN_5L_Model, self).__init__()
        self.layer1 = nn.Sequential(nn.Conv2d(3, 6, 5),nn.ReLU(True),nn.MaxPool2d(kern
        self.layer2 = nn.Sequential(nn.Conv2d(6, 16, 5),nn.ReLU(True),nn.MaxPool2d(ker
        self.layer3 = nn.Sequential(nn.Linear(16*5*5, 64),nn.ReLU(True))
        self.layer4 = nn.Sequential(nn.Linear(64, 32),nn.ReLU(True))
        self.layer5 = nn.Sequential(nn.Linear(32, 10))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.view(x.size()[0], -1)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        return x

device = torch.device("cuda")
Model = CNN_5L_Model().to(device)
summary(Model, input_size=(3,32,32))

# CNN with 4 Layers
class CNN_4L_Model(nn.Module):
    def __init__(self):
        super(CNN_4L_Model, self).__init__()
        self.layer1 = nn.Sequential(nn.Conv2d(3, 16, 5),nn.ReLU(True),nn.MaxPool2d(ker
        self.layer2 = nn.Sequential(nn.Linear(16*9*9, 64),nn.ReLU(True))
        self.layer3 = nn.Sequential(nn.Linear(64, 32),nn.ReLU(True))
```

```
        self.layer4 = nn.Sequential(nn.Linear(32, 10))
    def forward(self, x):
        x = self.layer1(x)
        x = x.view(x.size()[0], -1)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return x

device = torch.device("cuda")
Model = CNN_4L_Model().to(device)
summary(Model, input_size=(3,32,32))


class DNN_MNIST(nn.Module):
    def __init__(self, in_dim, hidden_1, hidden_2, hidden_3, out_dim):
        super(DNN_MNIST, self).__init__()
        self.layer1 = nn.Sequential(nn.Linear(in_dim, hidden_1),nn.BatchNorm1d(hidden_
        self.layer2 = nn.Sequential(nn.Linear(hidden_1, hidden_2),nn.BatchNorm1d(hidde
        self.layer3 = nn.Sequential(nn.Linear(hidden_2, hidden_3),nn.BatchNorm1d(hidde
        self.layer4 = nn.Sequential(nn.Linear(hidden_3, out_dim))
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return x
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 10, 30, 30]             280
              ReLU-2           [-1, 10, 30, 30]               0
         MaxPool2d-3           [-1, 10, 15, 15]               0
            Conv2d-4           [-1, 16, 13, 13]           1,456
              ReLU-5           [-1, 16, 13, 13]               0
         MaxPool2d-6             [-1, 16, 6, 6]               0
            Conv2d-7             [-1, 32, 4, 4]           4,640
              ReLU-8             [-1, 32, 4, 4]               0
            Linear-9                  [-1, 128]          65,664
             ReLU-10                  [-1, 128]               0
           Linear-11                   [-1, 32]           4,128
             ReLU-12                   [-1, 32]               0
           Linear-13                   [-1, 10]             330
================================================================
Total params: 76,498
Trainable params: 76,498
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.21
Params size (MB): 0.29
Estimated Total Size (MB): 0.51
----------------------------------------------------------------
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
              ReLU-2            [-1, 6, 28, 28]               0
         MaxPool2d-3            [-1, 6, 14, 14]               0
            Conv2d-4           [-1, 16, 10, 10]           2,416
              ReLU-5           [-1, 16, 10, 10]               0
         MaxPool2d-6             [-1, 16, 5, 5]               0
            Linear-7                   [-1, 64]          25,664
              ReLU-8                   [-1, 64]               0
            Linear-9                   [-1, 32]           2,080
             ReLU-10                   [-1, 32]               0
           Linear-11                   [-1, 10]             330
================================================================
Total params: 30,946
Trainable params: 30,946
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.11
Params size (MB): 0.12
Estimated Total Size (MB): 0.24
----------------------------------------------------------------
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 28, 28]           1,216
              ReLU-2           [-1, 16, 28, 28]               0
         MaxPool2d-3             [-1, 16, 9, 9]               0
            Linear-4                   [-1, 64]          83,008
              ReLU-5                   [-1, 64]               0
            Linear-6                   [-1, 32]           2,080
              ReLU-7                   [-1, 32]               0
```

```
            Linear-8                   [-1, 10]             330
================================================================
Total params: 86,634
Trainable params: 86,634
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.20
Params size (MB): 0.33
Estimated Total Size (MB): 0.55
----------------------------------------------------------------
```

6. CNN for CIFAR10 Dataset

In [19]:

```
# Define train function
def train_CIFAR(model_name,
                Epochs = 100,
                Batch  = 2000,
                Data_workers = 0,
                LR = 0.1):

# Initiate data
    transform = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,0.
    trainset = torchvision.datasets.CIFAR10(root='./data/',train=True,download=True,tr
    testset = torchvision.datasets.CIFAR10(root='./data/',train=False,download=True,tr
    trainloader = DataLoader(trainset, batch_size=Batch, shuffle=True, num_workers=Dat
    testloader  = DataLoader(testset,  batch_size=Batch, shuffle=True, num_workers=Dat
    print(trainset.classes)
    print(trainset.data.shape)
    print(testset.data.shape)

# Initiate model
    torch.cuda.is_available()
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    Model = model_name().to(device)

# loss & optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(Model.parameters(), lr=LR, momentum=0.9)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size = 5, gamma = 0.8)

# Training
    trainloss_list = []
    testloss_list  = []
    accuracy_list  = []
    lr_list = []

    for epoch in range(Epochs):
        Model.train()
        train_loss = 0.0
        for i, data in enumerate(trainloader):
            images, labels = data
            images = images.to(device)
            labels = labels.to(device)
            outputs = Model(images)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
```

```
            optimizer.step()
            train_loss += loss.item()
            total = (i+1)*Batch

# Evaluating
        Model.eval()
        with torch.no_grad():
            test_loss = 0
            correct = 0
            total = 0
            for data in testloader:
                images, labels = data
                images = images.to(device)
                labels = labels.to(device)
                outputs = Model(images)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
                _, pred = torch.max(outputs.data, 1)
                correct += (pred == labels).cpu().sum()
                total += labels.size(0)
            total = len(testloader.dataset)
            accuracy = 100.0*correct/total

# Save loss
        scheduler.step()
        lr_list.append(optimizer.state_dict()['param_groups'][0]['lr'])
        trainloss_list.append(train_loss)
        testloss_list.append(test_loss)
        accuracy_list.append(accuracy)
        print('{}/{} Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%) lr={}'.
                epoch, Epochs,test_loss, correct, total, accuracy, lr_list[-1]))

    return [trainloss_list,
            testloss_list,
            accuracy_list,
            lr_list]
```

In [20]:

```
[trainloss_CIFAR_6L, testloss_CIFAR_6L, accuracy_CIFAR_6L, lr_CIFAR_6L] = train_CIFAR(
[trainloss_CIFAR_5L, testloss_CIFAR_5L, accuracy_CIFAR_5L, lr_CIFAR_5L] = train_CIFAR(
[trainloss_CIFAR_4L, testloss_CIFAR_4L, accuracy_CIFAR_4L, lr_CIFAR_4L] = train_CIFAR(
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-1
0-python.tar.gz
  0%|          | 0/170498071 [00:00<?, ?it/s]
```

```
Extracting ./data/cifar-10-python.tar.gz to ./data/
Files already downloaded and verified
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tr
uck']
(50000, 32, 32, 3)
(10000, 32, 32, 3)
0/100 Test set: Average loss: 11.4972, Accuracy: 1196/10000 (11.96%) lr=0.1
1/100 Test set: Average loss: 10.9948, Accuracy: 2017/10000 (20.17%) lr=0.1
2/100 Test set: Average loss: 9.8133, Accuracy: 2814/10000 (28.14%) lr=0.1
3/100 Test set: Average loss: 12.1110, Accuracy: 2432/10000 (24.32%) lr=0.1
4/100 Test set: Average loss: 8.9427, Accuracy: 3513/10000 (35.13%) lr=0.080000000000
00002
5/100 Test set: Average loss: 7.8767, Accuracy: 4259/10000 (42.59%) lr=0.080000000000
00002
6/100 Test set: Average loss: 7.3213, Accuracy: 4692/10000 (46.92%) lr=0.080000000000
00002
7/100 Test set: Average loss: 7.1701, Accuracy: 4801/10000 (48.01%) lr=0.080000000000
00002
8/100 Test set: Average loss: 7.0645, Accuracy: 4906/10000 (49.06%) lr=0.080000000000
00002
9/100 Test set: Average loss: 6.4852, Accuracy: 5332/10000 (53.32%) lr=0.064000000000
00002
10/100 Test set: Average loss: 6.1200, Accuracy: 5595/10000 (55.95%) lr=0.06400000000
000002
11/100 Test set: Average loss: 6.1490, Accuracy: 5599/10000 (55.99%) lr=0.06400000000
000002
12/100 Test set: Average loss: 5.9511, Accuracy: 5743/10000 (57.43%) lr=0.06400000000
000002
13/100 Test set: Average loss: 6.0537, Accuracy: 5671/10000 (56.71%) lr=0.06400000000
000002
14/100 Test set: Average loss: 5.7531, Accuracy: 5952/10000 (59.52%) lr=0.05120000000
0000016
15/100 Test set: Average loss: 5.5666, Accuracy: 6048/10000 (60.48%) lr=0.05120000000
0000016
16/100 Test set: Average loss: 5.7908, Accuracy: 5869/10000 (58.69%) lr=0.05120000000
0000016
17/100 Test set: Average loss: 5.4143, Accuracy: 6205/10000 (62.05%) lr=0.05120000000
0000016
18/100 Test set: Average loss: 5.4567, Accuracy: 6179/10000 (61.79%) lr=0.05120000000
0000016
19/100 Test set: Average loss: 5.3410, Accuracy: 6264/10000 (62.64%) lr=0.04096000000
000002
20/100 Test set: Average loss: 5.3605, Accuracy: 6237/10000 (62.37%) lr=0.04096000000
000002
21/100 Test set: Average loss: 5.1835, Accuracy: 6397/10000 (63.97%) lr=0.04096000000
000002
22/100 Test set: Average loss: 5.1361, Accuracy: 6427/10000 (64.27%) lr=0.04096000000
000002
23/100 Test set: Average loss: 5.3670, Accuracy: 6248/10000 (62.48%) lr=0.04096000000
000002
24/100 Test set: Average loss: 5.1918, Accuracy: 6408/10000 (64.08%) lr=0.03276800000
000001
25/100 Test set: Average loss: 5.0312, Accuracy: 6549/10000 (65.49%) lr=0.03276800000
000001
26/100 Test set: Average loss: 5.0939, Accuracy: 6507/10000 (65.07%) lr=0.03276800000
000001
27/100 Test set: Average loss: 5.0018, Accuracy: 6595/10000 (65.95%) lr=0.03276800000
000001
28/100 Test set: Average loss: 5.1332, Accuracy: 6533/10000 (65.33%) lr=0.03276800000
000001
```

```
29/100 Test set: Average loss: 5.1078, Accuracy: 6538/10000 (65.38%) lr=0.02621440000
0000013
30/100 Test set: Average loss: 5.0218, Accuracy: 6624/10000 (66.24%) lr=0.02621440000
0000013
31/100 Test set: Average loss: 4.9955, Accuracy: 6657/10000 (66.57%) lr=0.02621440000
0000013
32/100 Test set: Average loss: 4.9905, Accuracy: 6638/10000 (66.38%) lr=0.02621440000
0000013
33/100 Test set: Average loss: 5.1347, Accuracy: 6600/10000 (66.00%) lr=0.02621440000
0000013
34/100 Test set: Average loss: 5.0687, Accuracy: 6641/10000 (66.41%) lr=0.02097152000
000001
35/100 Test set: Average loss: 5.1833, Accuracy: 6592/10000 (65.92%) lr=0.02097152000
000001
36/100 Test set: Average loss: 5.0821, Accuracy: 6694/10000 (66.94%) lr=0.02097152000
000001
37/100 Test set: Average loss: 5.0705, Accuracy: 6644/10000 (66.44%) lr=0.02097152000
000001
38/100 Test set: Average loss: 5.1292, Accuracy: 6670/10000 (66.70%) lr=0.02097152000
000001
39/100 Test set: Average loss: 5.1215, Accuracy: 6685/10000 (66.85%) lr=0.01677721600
0000008
40/100 Test set: Average loss: 5.1660, Accuracy: 6645/10000 (66.45%) lr=0.01677721600
0000008
41/100 Test set: Average loss: 5.3312, Accuracy: 6640/10000 (66.40%) lr=0.01677721600
0000008
42/100 Test set: Average loss: 5.3049, Accuracy: 6663/10000 (66.63%) lr=0.01677721600
0000008
43/100 Test set: Average loss: 5.3312, Accuracy: 6614/10000 (66.14%) lr=0.01677721600
0000008
44/100 Test set: Average loss: 5.3164, Accuracy: 6703/10000 (67.03%) lr=0.01342177280
0000007
45/100 Test set: Average loss: 5.3607, Accuracy: 6666/10000 (66.66%) lr=0.01342177280
0000007
46/100 Test set: Average loss: 5.3705, Accuracy: 6687/10000 (66.87%) lr=0.01342177280
0000007
47/100 Test set: Average loss: 5.4045, Accuracy: 6676/10000 (66.76%) lr=0.01342177280
0000007
48/100 Test set: Average loss: 5.4374, Accuracy: 6695/10000 (66.95%) lr=0.01342177280
0000007
49/100 Test set: Average loss: 5.4971, Accuracy: 6674/10000 (66.74%) lr=0.01073741824
0000006
50/100 Test set: Average loss: 5.5335, Accuracy: 6685/10000 (66.85%) lr=0.01073741824
0000006
51/100 Test set: Average loss: 5.5693, Accuracy: 6688/10000 (66.88%) lr=0.01073741824
0000006
52/100 Test set: Average loss: 5.6012, Accuracy: 6680/10000 (66.80%) lr=0.01073741824
0000006
53/100 Test set: Average loss: 5.7126, Accuracy: 6605/10000 (66.05%) lr=0.01073741824
0000006
54/100 Test set: Average loss: 5.8487, Accuracy: 6600/10000 (66.00%) lr=0.00858993459
2000005
55/100 Test set: Average loss: 5.7164, Accuracy: 6686/10000 (66.86%) lr=0.00858993459
2000005
56/100 Test set: Average loss: 5.7765, Accuracy: 6665/10000 (66.65%) lr=0.00858993459
2000005
57/100 Test set: Average loss: 5.8199, Accuracy: 6642/10000 (66.42%) lr=0.00858993459
2000005
58/100 Test set: Average loss: 5.8205, Accuracy: 6677/10000 (66.77%) lr=0.00858993459
2000005
```

```
59/100 Test set: Average loss: 5.9331, Accuracy: 6641/10000 (66.41%) lr=0.00687194767
36000045
60/100 Test set: Average loss: 5.9285, Accuracy: 6675/10000 (66.75%) lr=0.00687194767
36000045
61/100 Test set: Average loss: 5.9899, Accuracy: 6663/10000 (66.63%) lr=0.00687194767
36000045
62/100 Test set: Average loss: 6.0456, Accuracy: 6639/10000 (66.39%) lr=0.00687194767
36000045
63/100 Test set: Average loss: 6.0843, Accuracy: 6667/10000 (66.67%) lr=0.00687194767
36000045
64/100 Test set: Average loss: 6.1621, Accuracy: 6591/10000 (65.91%) lr=0.00549755813
8880004
65/100 Test set: Average loss: 6.1799, Accuracy: 6627/10000 (66.27%) lr=0.00549755813
8880004
66/100 Test set: Average loss: 6.1542, Accuracy: 6609/10000 (66.09%) lr=0.00549755813
8880004
67/100 Test set: Average loss: 6.1971, Accuracy: 6622/10000 (66.22%) lr=0.00549755813
8880004
68/100 Test set: Average loss: 6.2387, Accuracy: 6613/10000 (66.13%) lr=0.00549755813
8880004
69/100 Test set: Average loss: 6.2818, Accuracy: 6626/10000 (66.26%) lr=0.00439804651
1104004
70/100 Test set: Average loss: 6.3277, Accuracy: 6617/10000 (66.17%) lr=0.00439804651
1104004
71/100 Test set: Average loss: 6.3471, Accuracy: 6597/10000 (65.97%) lr=0.00439804651
1104004
72/100 Test set: Average loss: 6.4176, Accuracy: 6625/10000 (66.25%) lr=0.00439804651
1104004
73/100 Test set: Average loss: 6.4100, Accuracy: 6593/10000 (65.93%) lr=0.00439804651
1104004
74/100 Test set: Average loss: 6.4598, Accuracy: 6605/10000 (66.05%) lr=0.00351843720
88832034
75/100 Test set: Average loss: 6.4762, Accuracy: 6622/10000 (66.22%) lr=0.00351843720
88832034
76/100 Test set: Average loss: 6.5156, Accuracy: 6606/10000 (66.06%) lr=0.00351843720
88832034
77/100 Test set: Average loss: 6.5404, Accuracy: 6597/10000 (65.97%) lr=0.00351843720
88832034
78/100 Test set: Average loss: 6.5759, Accuracy: 6599/10000 (65.99%) lr=0.00351843720
88832034
79/100 Test set: Average loss: 6.5961, Accuracy: 6615/10000 (66.15%) lr=0.00281474976
7106563
80/100 Test set: Average loss: 6.6373, Accuracy: 6606/10000 (66.06%) lr=0.00281474976
7106563
81/100 Test set: Average loss: 6.6508, Accuracy: 6602/10000 (66.02%) lr=0.00281474976
7106563
82/100 Test set: Average loss: 6.6771, Accuracy: 6570/10000 (65.70%) lr=0.00281474976
7106563
83/100 Test set: Average loss: 6.6830, Accuracy: 6593/10000 (65.93%) lr=0.00281474976
7106563
84/100 Test set: Average loss: 6.7283, Accuracy: 6591/10000 (65.91%) lr=0.00225179981
36852503
85/100 Test set: Average loss: 6.7377, Accuracy: 6600/10000 (66.00%) lr=0.00225179981
36852503
86/100 Test set: Average loss: 6.7651, Accuracy: 6584/10000 (65.84%) lr=0.00225179981
36852503
87/100 Test set: Average loss: 6.7717, Accuracy: 6579/10000 (65.79%) lr=0.00225179981
36852503
88/100 Test set: Average loss: 6.8119, Accuracy: 6569/10000 (65.69%) lr=0.00225179981
36852503
```

```
89/100 Test set: Average loss: 6.8313, Accuracy: 6557/10000 (65.57%) lr=0.00180143985
09482003
90/100 Test set: Average loss: 6.8446, Accuracy: 6601/10000 (66.01%) lr=0.00180143985
09482003
91/100 Test set: Average loss: 6.8757, Accuracy: 6575/10000 (65.75%) lr=0.00180143985
09482003
92/100 Test set: Average loss: 6.8702, Accuracy: 6585/10000 (65.85%) lr=0.00180143985
09482003
93/100 Test set: Average loss: 6.9061, Accuracy: 6570/10000 (65.70%) lr=0.00180143985
09482003
94/100 Test set: Average loss: 6.9033, Accuracy: 6582/10000 (65.82%) lr=0.00144115188
07585604
95/100 Test set: Average loss: 6.9299, Accuracy: 6586/10000 (65.86%) lr=0.00144115188
07585604
96/100 Test set: Average loss: 6.9452, Accuracy: 6589/10000 (65.89%) lr=0.00144115188
07585604
97/100 Test set: Average loss: 6.9612, Accuracy: 6591/10000 (65.91%) lr=0.00144115188
07585604
98/100 Test set: Average loss: 6.9690, Accuracy: 6588/10000 (65.88%) lr=0.00144115188
07585604
99/100 Test set: Average loss: 6.9831, Accuracy: 6574/10000 (65.74%) lr=0.00115292150
46068484
Files already downloaded and verified
Files already downloaded and verified
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tr
uck']
(50000, 32, 32, 3)
(10000, 32, 32, 3)
0/100 Test set: Average loss: 11.3761, Accuracy: 1638/10000 (16.38%) lr=0.1
1/100 Test set: Average loss: 9.5720, Accuracy: 3072/10000 (30.72%) lr=0.1
2/100 Test set: Average loss: 8.5533, Accuracy: 3740/10000 (37.40%) lr=0.1
3/100 Test set: Average loss: 7.8275, Accuracy: 4346/10000 (43.46%) lr=0.1
4/100 Test set: Average loss: 7.2216, Accuracy: 4728/10000 (47.28%) lr=0.080000000000
00002
5/100 Test set: Average loss: 6.9003, Accuracy: 4987/10000 (49.87%) lr=0.080000000000
00002
6/100 Test set: Average loss: 6.8352, Accuracy: 5066/10000 (50.66%) lr=0.080000000000
00002
7/100 Test set: Average loss: 6.5005, Accuracy: 5290/10000 (52.90%) lr=0.080000000000
00002
8/100 Test set: Average loss: 6.3710, Accuracy: 5458/10000 (54.58%) lr=0.080000000000
00002
9/100 Test set: Average loss: 6.2487, Accuracy: 5570/10000 (55.70%) lr=0.064000000000
00002
10/100 Test set: Average loss: 6.2744, Accuracy: 5576/10000 (55.76%) lr=0.06400000000
000002
11/100 Test set: Average loss: 5.8807, Accuracy: 5804/10000 (58.04%) lr=0.06400000000
000002
12/100 Test set: Average loss: 5.8761, Accuracy: 5848/10000 (58.48%) lr=0.06400000000
000002
13/100 Test set: Average loss: 5.8112, Accuracy: 5949/10000 (59.49%) lr=0.06400000000
000002
14/100 Test set: Average loss: 5.8934, Accuracy: 5910/10000 (59.10%) lr=0.05120000000
0000016
15/100 Test set: Average loss: 5.6268, Accuracy: 6075/10000 (60.75%) lr=0.05120000000
0000016
16/100 Test set: Average loss: 5.6757, Accuracy: 6019/10000 (60.19%) lr=0.05120000000
0000016
17/100 Test set: Average loss: 5.7076, Accuracy: 5999/10000 (59.99%) lr=0.05120000000
0000016
```

```
18/100 Test set: Average loss: 5.5963, Accuracy: 6084/10000 (60.84%) lr=0.05120000000
0000016
19/100 Test set: Average loss: 5.5842, Accuracy: 6044/10000 (60.44%) lr=0.04096000000
000002
20/100 Test set: Average loss: 5.4520, Accuracy: 6167/10000 (61.67%) lr=0.04096000000
000002
21/100 Test set: Average loss: 5.4679, Accuracy: 6181/10000 (61.81%) lr=0.04096000000
000002
22/100 Test set: Average loss: 5.4854, Accuracy: 6188/10000 (61.88%) lr=0.04096000000
000002
23/100 Test set: Average loss: 5.4526, Accuracy: 6266/10000 (62.66%) lr=0.04096000000
000002
24/100 Test set: Average loss: 5.4008, Accuracy: 6198/10000 (61.98%) lr=0.03276800000
000001
25/100 Test set: Average loss: 5.3810, Accuracy: 6281/10000 (62.81%) lr=0.03276800000
000001
26/100 Test set: Average loss: 5.3342, Accuracy: 6287/10000 (62.87%) lr=0.03276800000
000001
27/100 Test set: Average loss: 5.3875, Accuracy: 6289/10000 (62.89%) lr=0.03276800000
000001
28/100 Test set: Average loss: 5.4997, Accuracy: 6241/10000 (62.41%) lr=0.03276800000
000001
29/100 Test set: Average loss: 5.4310, Accuracy: 6313/10000 (63.13%) lr=0.02621440000
0000013
30/100 Test set: Average loss: 5.4305, Accuracy: 6285/10000 (62.85%) lr=0.02621440000
0000013
31/100 Test set: Average loss: 5.3832, Accuracy: 6338/10000 (63.38%) lr=0.02621440000
0000013
32/100 Test set: Average loss: 5.3811, Accuracy: 6361/10000 (63.61%) lr=0.02621440000
0000013
33/100 Test set: Average loss: 5.3673, Accuracy: 6363/10000 (63.63%) lr=0.02621440000
0000013
34/100 Test set: Average loss: 5.4029, Accuracy: 6333/10000 (63.33%) lr=0.02097152000
000001
35/100 Test set: Average loss: 5.4477, Accuracy: 6357/10000 (63.57%) lr=0.02097152000
000001
36/100 Test set: Average loss: 5.5253, Accuracy: 6346/10000 (63.46%) lr=0.02097152000
000001
37/100 Test set: Average loss: 5.6199, Accuracy: 6280/10000 (62.80%) lr=0.02097152000
000001
38/100 Test set: Average loss: 5.4762, Accuracy: 6343/10000 (63.43%) lr=0.02097152000
000001
39/100 Test set: Average loss: 5.4804, Accuracy: 6330/10000 (63.30%) lr=0.01677721600
0000008
40/100 Test set: Average loss: 5.4709, Accuracy: 6322/10000 (63.22%) lr=0.01677721600
0000008
41/100 Test set: Average loss: 5.5115, Accuracy: 6311/10000 (63.11%) lr=0.01677721600
0000008
42/100 Test set: Average loss: 5.5936, Accuracy: 6298/10000 (62.98%) lr=0.01677721600
0000008
43/100 Test set: Average loss: 5.5410, Accuracy: 6331/10000 (63.31%) lr=0.01677721600
0000008
44/100 Test set: Average loss: 5.4903, Accuracy: 6345/10000 (63.45%) lr=0.01342177280
0000007
45/100 Test set: Average loss: 5.5418, Accuracy: 6277/10000 (62.77%) lr=0.01342177280
0000007
46/100 Test set: Average loss: 5.5700, Accuracy: 6326/10000 (63.26%) lr=0.01342177280
0000007
47/100 Test set: Average loss: 5.6132, Accuracy: 6362/10000 (63.62%) lr=0.01342177280
0000007
```

```
48/100 Test set: Average loss: 5.6210, Accuracy: 6319/10000 (63.19%) lr=0.01342177280
0000007
49/100 Test set: Average loss: 5.6273, Accuracy: 6316/10000 (63.16%) lr=0.01073741824
0000006
50/100 Test set: Average loss: 5.5958, Accuracy: 6354/10000 (63.54%) lr=0.01073741824
0000006
51/100 Test set: Average loss: 5.6066, Accuracy: 6352/10000 (63.52%) lr=0.01073741824
0000006
52/100 Test set: Average loss: 5.6736, Accuracy: 6330/10000 (63.30%) lr=0.01073741824
0000006
53/100 Test set: Average loss: 5.6650, Accuracy: 6337/10000 (63.37%) lr=0.01073741824
0000006
54/100 Test set: Average loss: 5.6509, Accuracy: 6311/10000 (63.11%) lr=0.00858993459
2000005
55/100 Test set: Average loss: 5.6729, Accuracy: 6356/10000 (63.56%) lr=0.00858993459
2000005
56/100 Test set: Average loss: 5.7058, Accuracy: 6334/10000 (63.34%) lr=0.00858993459
2000005
57/100 Test set: Average loss: 5.7096, Accuracy: 6321/10000 (63.21%) lr=0.00858993459
2000005
58/100 Test set: Average loss: 5.7171, Accuracy: 6332/10000 (63.32%) lr=0.00858993459
2000005
59/100 Test set: Average loss: 5.6961, Accuracy: 6330/10000 (63.30%) lr=0.00687194767
36000045
60/100 Test set: Average loss: 5.7364, Accuracy: 6328/10000 (63.28%) lr=0.00687194767
36000045
61/100 Test set: Average loss: 5.7784, Accuracy: 6310/10000 (63.10%) lr=0.00687194767
36000045
62/100 Test set: Average loss: 5.7908, Accuracy: 6318/10000 (63.18%) lr=0.00687194767
36000045
63/100 Test set: Average loss: 5.7636, Accuracy: 6333/10000 (63.33%) lr=0.00687194767
36000045
64/100 Test set: Average loss: 5.7670, Accuracy: 6337/10000 (63.37%) lr=0.00549755813
8880004
65/100 Test set: Average loss: 5.7683, Accuracy: 6341/10000 (63.41%) lr=0.00549755813
8880004
66/100 Test set: Average loss: 5.8089, Accuracy: 6331/10000 (63.31%) lr=0.00549755813
8880004
67/100 Test set: Average loss: 5.8020, Accuracy: 6311/10000 (63.11%) lr=0.00549755813
8880004
68/100 Test set: Average loss: 5.7959, Accuracy: 6331/10000 (63.31%) lr=0.00549755813
8880004
69/100 Test set: Average loss: 5.8202, Accuracy: 6327/10000 (63.27%) lr=0.00439804651
1104004
70/100 Test set: Average loss: 5.8419, Accuracy: 6295/10000 (62.95%) lr=0.00439804651
1104004
71/100 Test set: Average loss: 5.8292, Accuracy: 6319/10000 (63.19%) lr=0.00439804651
1104004
72/100 Test set: Average loss: 5.8241, Accuracy: 6327/10000 (63.27%) lr=0.00439804651
1104004
73/100 Test set: Average loss: 5.8243, Accuracy: 6331/10000 (63.31%) lr=0.00439804651
1104004
74/100 Test set: Average loss: 5.8772, Accuracy: 6316/10000 (63.16%) lr=0.00351843720
88832034
75/100 Test set: Average loss: 5.8491, Accuracy: 6324/10000 (63.24%) lr=0.00351843720
88832034
76/100 Test set: Average loss: 5.8581, Accuracy: 6327/10000 (63.27%) lr=0.00351843720
88832034
77/100 Test set: Average loss: 5.8528, Accuracy: 6317/10000 (63.17%) lr=0.00351843720
88832034
```

```
78/100 Test set: Average loss: 5.8723, Accuracy: 6315/10000 (63.15%) lr=0.00351843720
88832034
79/100 Test set: Average loss: 5.8660, Accuracy: 6321/10000 (63.21%) lr=0.00281474976
7106563
80/100 Test set: Average loss: 5.8707, Accuracy: 6319/10000 (63.19%) lr=0.00281474976
7106563
81/100 Test set: Average loss: 5.8817, Accuracy: 6335/10000 (63.35%) lr=0.00281474976
7106563
82/100 Test set: Average loss: 5.9155, Accuracy: 6319/10000 (63.19%) lr=0.00281474976
7106563
83/100 Test set: Average loss: 5.9057, Accuracy: 6314/10000 (63.14%) lr=0.00281474976
7106563
84/100 Test set: Average loss: 5.9127, Accuracy: 6317/10000 (63.17%) lr=0.00225179981
36852503
85/100 Test set: Average loss: 5.9174, Accuracy: 6331/10000 (63.31%) lr=0.00225179981
36852503
86/100 Test set: Average loss: 5.9202, Accuracy: 6330/10000 (63.30%) lr=0.00225179981
36852503
87/100 Test set: Average loss: 5.9261, Accuracy: 6320/10000 (63.20%) lr=0.00225179981
36852503
88/100 Test set: Average loss: 5.9241, Accuracy: 6351/10000 (63.51%) lr=0.00225179981
36852503
89/100 Test set: Average loss: 5.9256, Accuracy: 6302/10000 (63.02%) lr=0.00180143985
09482003
90/100 Test set: Average loss: 5.9207, Accuracy: 6327/10000 (63.27%) lr=0.00180143985
09482003
91/100 Test set: Average loss: 5.9233, Accuracy: 6313/10000 (63.13%) lr=0.00180143985
09482003
92/100 Test set: Average loss: 5.9315, Accuracy: 6335/10000 (63.35%) lr=0.00180143985
09482003
93/100 Test set: Average loss: 5.9450, Accuracy: 6331/10000 (63.31%) lr=0.00180143985
09482003
94/100 Test set: Average loss: 5.9487, Accuracy: 6333/10000 (63.33%) lr=0.00144115188
07585604
95/100 Test set: Average loss: 5.9364, Accuracy: 6320/10000 (63.20%) lr=0.00144115188
07585604
96/100 Test set: Average loss: 5.9433, Accuracy: 6324/10000 (63.24%) lr=0.00144115188
07585604
97/100 Test set: Average loss: 5.9429, Accuracy: 6319/10000 (63.19%) lr=0.00144115188
07585604
98/100 Test set: Average loss: 5.9597, Accuracy: 6348/10000 (63.48%) lr=0.00144115188
07585604
99/100 Test set: Average loss: 5.9632, Accuracy: 6332/10000 (63.32%) lr=0.00115292150
46068484
Files already downloaded and verified
Files already downloaded and verified
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tr
uck']
(50000, 32, 32, 3)
(10000, 32, 32, 3)
0/100 Test set: Average loss: 9.8120, Accuracy: 2754/10000 (27.54%) lr=0.1
1/100 Test set: Average loss: 7.9896, Accuracy: 4115/10000 (41.15%) lr=0.1
2/100 Test set: Average loss: 7.2923, Accuracy: 4648/10000 (46.48%) lr=0.1
3/100 Test set: Average loss: 6.5373, Accuracy: 5280/10000 (52.80%) lr=0.1
4/100 Test set: Average loss: 6.2894, Accuracy: 5511/10000 (55.11%) lr=0.080000000000
00002
5/100 Test set: Average loss: 6.1818, Accuracy: 5607/10000 (56.07%) lr=0.080000000000
00002
6/100 Test set: Average loss: 5.7095, Accuracy: 5992/10000 (59.92%) lr=0.080000000000
00002
```

```
7/100 Test set: Average loss: 6.0366, Accuracy: 5750/10000 (57.50%) lr=0.080000000000
00002
8/100 Test set: Average loss: 5.4130, Accuracy: 6234/10000 (62.34%) lr=0.080000000000
00002
9/100 Test set: Average loss: 5.2049, Accuracy: 6378/10000 (63.78%) lr=0.064000000000
00002
10/100 Test set: Average loss: 5.1428, Accuracy: 6436/10000 (64.36%) lr=0.06400000000
000002
11/100 Test set: Average loss: 5.2607, Accuracy: 6411/10000 (64.11%) lr=0.06400000000
000002
12/100 Test set: Average loss: 5.1775, Accuracy: 6483/10000 (64.83%) lr=0.06400000000
000002
13/100 Test set: Average loss: 5.0603, Accuracy: 6576/10000 (65.76%) lr=0.06400000000
000002
14/100 Test set: Average loss: 5.0856, Accuracy: 6528/10000 (65.28%) lr=0.05120000000
0000016
15/100 Test set: Average loss: 5.0196, Accuracy: 6604/10000 (66.04%) lr=0.05120000000
0000016
16/100 Test set: Average loss: 4.9759, Accuracy: 6664/10000 (66.64%) lr=0.05120000000
0000016
17/100 Test set: Average loss: 5.0205, Accuracy: 6594/10000 (65.94%) lr=0.05120000000
0000016
18/100 Test set: Average loss: 5.1926, Accuracy: 6557/10000 (65.57%) lr=0.05120000000
0000016
19/100 Test set: Average loss: 5.0766, Accuracy: 6671/10000 (66.71%) lr=0.04096000000
000002
20/100 Test set: Average loss: 5.0745, Accuracy: 6675/10000 (66.75%) lr=0.04096000000
000002
21/100 Test set: Average loss: 5.1688, Accuracy: 6649/10000 (66.49%) lr=0.04096000000
000002
22/100 Test set: Average loss: 5.3057, Accuracy: 6593/10000 (65.93%) lr=0.04096000000
000002
23/100 Test set: Average loss: 5.2260, Accuracy: 6657/10000 (66.57%) lr=0.04096000000
000002
24/100 Test set: Average loss: 5.2761, Accuracy: 6615/10000 (66.15%) lr=0.03276800000
000001
25/100 Test set: Average loss: 5.2547, Accuracy: 6666/10000 (66.66%) lr=0.03276800000
000001
26/100 Test set: Average loss: 5.3979, Accuracy: 6664/10000 (66.64%) lr=0.03276800000
000001
27/100 Test set: Average loss: 5.4451, Accuracy: 6590/10000 (65.90%) lr=0.03276800000
000001
28/100 Test set: Average loss: 5.7002, Accuracy: 6527/10000 (65.27%) lr=0.03276800000
000001
29/100 Test set: Average loss: 5.6881, Accuracy: 6500/10000 (65.00%) lr=0.02621440000
0000013
30/100 Test set: Average loss: 5.6038, Accuracy: 6559/10000 (65.59%) lr=0.02621440000
0000013
31/100 Test set: Average loss: 5.6379, Accuracy: 6600/10000 (66.00%) lr=0.02621440000
0000013
32/100 Test set: Average loss: 5.7016, Accuracy: 6615/10000 (66.15%) lr=0.02621440000
0000013
33/100 Test set: Average loss: 5.7443, Accuracy: 6605/10000 (66.05%) lr=0.02621440000
0000013
34/100 Test set: Average loss: 5.8488, Accuracy: 6602/10000 (66.02%) lr=0.02097152000
000001
35/100 Test set: Average loss: 5.8713, Accuracy: 6559/10000 (65.59%) lr=0.02097152000
000001
36/100 Test set: Average loss: 5.9650, Accuracy: 6616/10000 (66.16%) lr=0.02097152000
000001
```

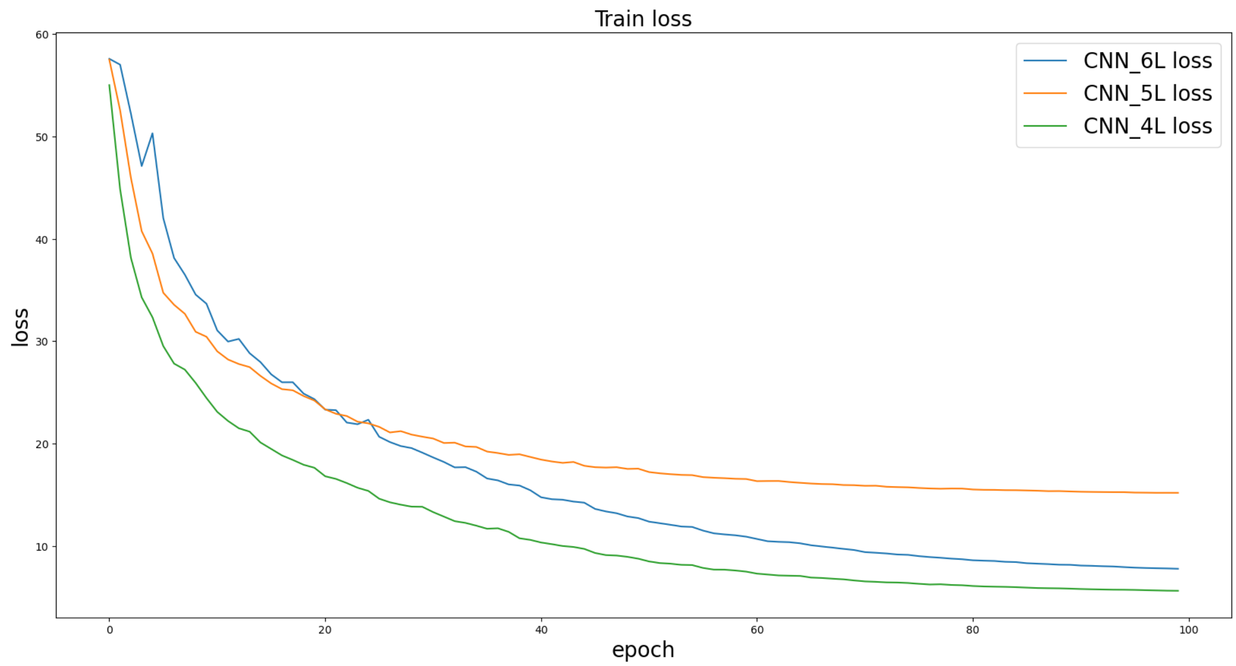
```
37/100 Test set: Average loss: 5.9474, Accuracy: 6591/10000 (65.91%) lr=0.02097152000
000001
38/100 Test set: Average loss: 6.0089, Accuracy: 6610/10000 (66.10%) lr=0.02097152000
000001
39/100 Test set: Average loss: 6.1164, Accuracy: 6561/10000 (65.61%) lr=0.01677721600
0000008
40/100 Test set: Average loss: 6.2020, Accuracy: 6582/10000 (65.82%) lr=0.01677721600
0000008
41/100 Test set: Average loss: 6.2185, Accuracy: 6563/10000 (65.63%) lr=0.01677721600
0000008
42/100 Test set: Average loss: 6.2845, Accuracy: 6561/10000 (65.61%) lr=0.01677721600
0000008
43/100 Test set: Average loss: 6.3287, Accuracy: 6605/10000 (66.05%) lr=0.01677721600
0000008
44/100 Test set: Average loss: 6.5299, Accuracy: 6533/10000 (65.33%) lr=0.01342177280
0000007
45/100 Test set: Average loss: 6.4611, Accuracy: 6581/10000 (65.81%) lr=0.01342177280
0000007
46/100 Test set: Average loss: 6.5387, Accuracy: 6542/10000 (65.42%) lr=0.01342177280
0000007
47/100 Test set: Average loss: 6.6165, Accuracy: 6549/10000 (65.49%) lr=0.01342177280
0000007
48/100 Test set: Average loss: 6.7161, Accuracy: 6542/10000 (65.42%) lr=0.01342177280
0000007
49/100 Test set: Average loss: 6.7060, Accuracy: 6547/10000 (65.47%) lr=0.01073741824
0000006
50/100 Test set: Average loss: 6.7391, Accuracy: 6530/10000 (65.30%) lr=0.01073741824
0000006
51/100 Test set: Average loss: 6.8305, Accuracy: 6547/10000 (65.47%) lr=0.01073741824
0000006
52/100 Test set: Average loss: 6.8790, Accuracy: 6549/10000 (65.49%) lr=0.01073741824
0000006
53/100 Test set: Average loss: 6.9958, Accuracy: 6559/10000 (65.59%) lr=0.01073741824
0000006
54/100 Test set: Average loss: 6.9702, Accuracy: 6540/10000 (65.40%) lr=0.00858993459
2000005
55/100 Test set: Average loss: 7.0333, Accuracy: 6547/10000 (65.47%) lr=0.00858993459
2000005
56/100 Test set: Average loss: 7.1210, Accuracy: 6545/10000 (65.45%) lr=0.00858993459
2000005
57/100 Test set: Average loss: 7.1305, Accuracy: 6566/10000 (65.66%) lr=0.00858993459
2000005
58/100 Test set: Average loss: 7.1925, Accuracy: 6527/10000 (65.27%) lr=0.00858993459
2000005
59/100 Test set: Average loss: 7.2458, Accuracy: 6535/10000 (65.35%) lr=0.00687194767
36000045
60/100 Test set: Average loss: 7.2863, Accuracy: 6529/10000 (65.29%) lr=0.00687194767
36000045
61/100 Test set: Average loss: 7.3414, Accuracy: 6522/10000 (65.22%) lr=0.00687194767
36000045
62/100 Test set: Average loss: 7.3729, Accuracy: 6532/10000 (65.32%) lr=0.00687194767
36000045
63/100 Test set: Average loss: 7.4513, Accuracy: 6537/10000 (65.37%) lr=0.00687194767
36000045
64/100 Test set: Average loss: 7.5043, Accuracy: 6532/10000 (65.32%) lr=0.00549755813
8880004
65/100 Test set: Average loss: 7.4863, Accuracy: 6543/10000 (65.43%) lr=0.00549755813
8880004
66/100 Test set: Average loss: 7.5301, Accuracy: 6537/10000 (65.37%) lr=0.00549755813
8880004
```

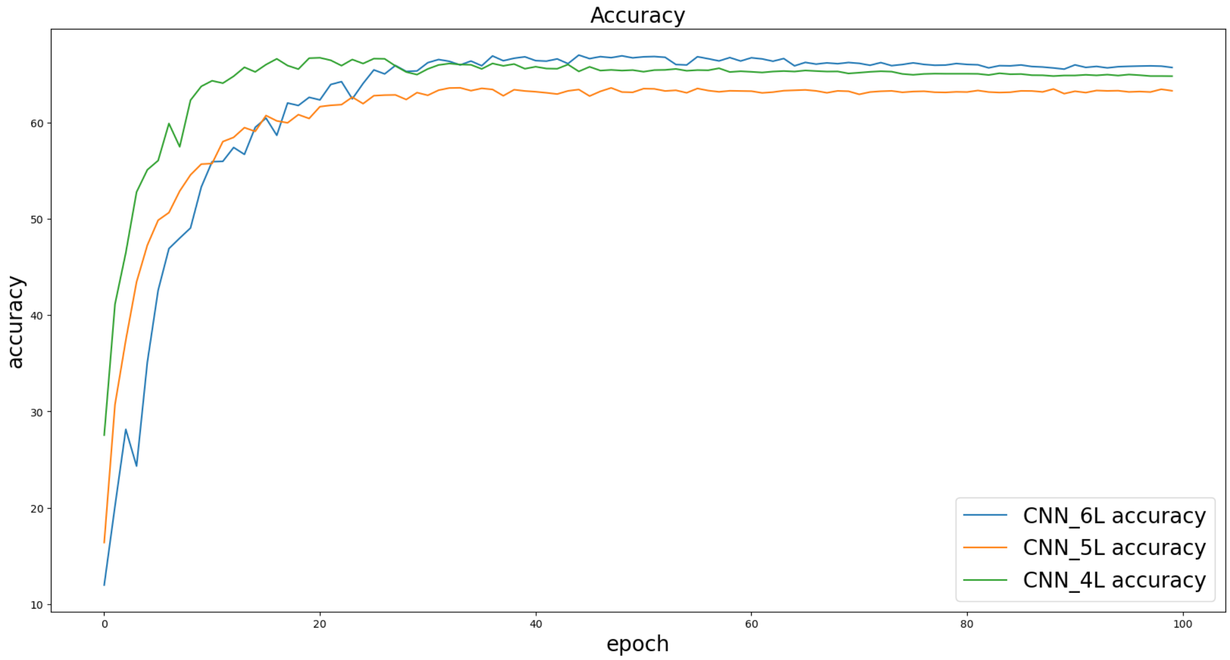
```
67/100 Test set: Average loss: 7.5544, Accuracy: 6532/10000 (65.32%) lr=0.00549755813
8880004
68/100 Test set: Average loss: 7.5899, Accuracy: 6533/10000 (65.33%) lr=0.00549755813
8880004
69/100 Test set: Average loss: 7.6593, Accuracy: 6512/10000 (65.12%) lr=0.00439804651
1104004
70/100 Test set: Average loss: 7.6902, Accuracy: 6520/10000 (65.20%) lr=0.00439804651
1104004
71/100 Test set: Average loss: 7.6875, Accuracy: 6529/10000 (65.29%) lr=0.00439804651
1104004
72/100 Test set: Average loss: 7.7629, Accuracy: 6535/10000 (65.35%) lr=0.00439804651
1104004
73/100 Test set: Average loss: 7.7807, Accuracy: 6531/10000 (65.31%) lr=0.00439804651
1104004
74/100 Test set: Average loss: 7.8073, Accuracy: 6507/10000 (65.07%) lr=0.00351843720
88832034
75/100 Test set: Average loss: 7.8239, Accuracy: 6498/10000 (64.98%) lr=0.00351843720
88832034
76/100 Test set: Average loss: 7.8512, Accuracy: 6507/10000 (65.07%) lr=0.00351843720
88832034
77/100 Test set: Average loss: 7.8950, Accuracy: 6510/10000 (65.10%) lr=0.00351843720
88832034
78/100 Test set: Average loss: 7.8904, Accuracy: 6509/10000 (65.09%) lr=0.00351843720
88832034
79/100 Test set: Average loss: 7.9370, Accuracy: 6509/10000 (65.09%) lr=0.00281474976
7106563
80/100 Test set: Average loss: 7.9653, Accuracy: 6509/10000 (65.09%) lr=0.00281474976
7106563
81/100 Test set: Average loss: 7.9570, Accuracy: 6508/10000 (65.08%) lr=0.00281474976
7106563
82/100 Test set: Average loss: 8.0170, Accuracy: 6496/10000 (64.96%) lr=0.00281474976
7106563
83/100 Test set: Average loss: 8.0152, Accuracy: 6514/10000 (65.14%) lr=0.00281474976
7106563
84/100 Test set: Average loss: 8.0460, Accuracy: 6504/10000 (65.04%) lr=0.00225179981
36852503
85/100 Test set: Average loss: 8.0598, Accuracy: 6506/10000 (65.06%) lr=0.00225179981
36852503
86/100 Test set: Average loss: 8.0640, Accuracy: 6494/10000 (64.94%) lr=0.00225179981
36852503
87/100 Test set: Average loss: 8.0820, Accuracy: 6493/10000 (64.93%) lr=0.00225179981
36852503
88/100 Test set: Average loss: 8.1047, Accuracy: 6485/10000 (64.85%) lr=0.00225179981
36852503
89/100 Test set: Average loss: 8.1275, Accuracy: 6491/10000 (64.91%) lr=0.00180143985
09482003
90/100 Test set: Average loss: 8.1269, Accuracy: 6491/10000 (64.91%) lr=0.00180143985
09482003
91/100 Test set: Average loss: 8.1480, Accuracy: 6498/10000 (64.98%) lr=0.00180143985
09482003
92/100 Test set: Average loss: 8.1685, Accuracy: 6492/10000 (64.92%) lr=0.00180143985
09482003
93/100 Test set: Average loss: 8.1839, Accuracy: 6500/10000 (65.00%) lr=0.00180143985
09482003
94/100 Test set: Average loss: 8.1884, Accuracy: 6490/10000 (64.90%) lr=0.00144115188
07585604
95/100 Test set: Average loss: 8.1905, Accuracy: 6501/10000 (65.01%) lr=0.00144115188
07585604
96/100 Test set: Average loss: 8.2088, Accuracy: 6494/10000 (64.94%) lr=0.00144115188
07585604
```

```
97/100 Test set: Average loss: 8.2175, Accuracy: 6485/10000 (64.85%) lr=0.00144115188
07585604
98/100 Test set: Average loss: 8.2321, Accuracy: 6485/10000 (64.85%) lr=0.00144115188
07585604
99/100 Test set: Average loss: 8.2471, Accuracy: 6484/10000 (64.84%) lr=0.00115292150
46068484
```

In [21]:

```
# Plot loss & acc
plt.figure(figsize=(20,10))
plt.plot(trainloss_CIFAR_6L, label='CNN_6L loss')
plt.plot(trainloss_CIFAR_5L, label='CNN_5L loss')
plt.plot(trainloss_CIFAR_4L, label='CNN_4L loss')
plt.xlabel('epoch',fontsize=20)
plt.ylabel('loss',fontsize=20)
plt.title('Train loss',fontsize=20)
plt.legend(fontsize=20)
plt.show()

plt.figure(figsize=(20,10))
plt.plot(accuracy_CIFAR_6L, label='CNN_6L accuracy')
plt.plot(accuracy_CIFAR_5L, label='CNN_5L accuracy')
plt.plot(accuracy_CIFAR_4L, label='CNN_4L accuracy')
plt.xlabel('epoch',fontsize=20)
plt.ylabel('accuracy',fontsize=20)
plt.title('Accuracy',fontsize=20)
plt.legend(fontsize=20)
plt.show()
```





7. DNN for MNIST Dataset

In [22]:

```python
# Define train function
def train_MNIST(model_name,
                Epochs = 50,
                Batch  = 2000,
                Data_workers = 0,
                LR = 0.1):
# Initiate data
    trainset = torchvision.datasets.MNIST(root='./data/',train=True,download=True,trar
    testset = torchvision.datasets.MNIST(root='./data/',train=False,download=True,trar
    trainloader = DataLoader(trainset, batch_size=Batch, shuffle=True, num_workers=Dat
    testloader  = DataLoader(testset,  batch_size=Batch, shuffle=True, num_workers=Dat
    print(trainset.classes)
    print(trainset.data.shape)
    print(testset.data.shape)

# Initiate model
    torch.cuda.is_available()
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    Model = model_name.to(device)

# loss & optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(Model.parameters(), lr=LR, momentum=0.9)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size = 5, gamma = 0.8)

# Training
    trainloss_list = []
    testloss_list  = []
    accuracy_list  = []
    lr_list = []
    for epoch in range(Epochs):
        Model.train()
        train_loss = 0.0
        for i, data in enumerate(trainloader):
            images, labels = data
```

```
            images = (images.view(-1, 28*28)).to(device)
            labels = labels.to(device)
            outputs = Model(images)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
            total = (i+1)*Batch

# Evaluating
        Model.eval()
        with torch.no_grad():
            test_loss = 0
            correct = 0
            total = 0
            for data in testloader:
                images, labels = data
                images = (images.view(-1, 28*28)).to(device)
                labels = labels.to(device)
                outputs = Model(images)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
                _, pred = torch.max(outputs.data, 1)
                correct += (pred == labels).cpu().sum()
                total += labels.size(0)
            total = len(testloader.dataset)
            accuracy = 100.0*correct/total

# Save loss
        scheduler.step()
        lr_list.append(optimizer.state_dict()['param_groups'][0]['lr'])
        trainloss_list.append(train_loss)
        testloss_list.append(test_loss)
        accuracy_list.append(accuracy)
        print('{}/{} Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%) lr={}'.
                epoch, Epochs,test_loss, correct, total, accuracy, lr_list[-1]))

    return [trainloss_list,
            testloss_list,
            accuracy_list,
            lr_list]
```

In [23]:

```
[trainloss_MNIST1, testloss_MNIST1, accuracy_MNIST1, lr_MNIST1] = train_MNIST(model_na
[trainloss_MNIST2, testloss_MNIST2, accuracy_MNIST2, lr_MNIST2] = train_MNIST(model_na
[trainloss_MNIST3, testloss_MNIST3, accuracy_MNIST3, lr_MNIST3] = train_MNIST(model_na
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNI
ST\raw\train-images-idx3-ubyte.gz
  0%|          | 0/9912422 [00:00<?, ?it/s]
Extracting ./data/MNIST\raw\train-images-idx3-ubyte.gz to ./data/MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNI
ST\raw\train-labels-idx1-ubyte.gz
  0%|          | 0/28881 [00:00<?, ?it/s]
```

```
Extracting ./data/MNIST\raw\train-labels-idx1-ubyte.gz to ./data/MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIS
T\raw\t10k-images-idx3-ubyte.gz
  0%|          | 0/1648877 [00:00<?, ?it/s]
Extracting ./data/MNIST\raw\t10k-images-idx3-ubyte.gz to ./data/MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIS
T\raw\t10k-labels-idx1-ubyte.gz
  0%|          | 0/4542 [00:00<?, ?it/s]
```

```
Extracting ./data/MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data/MNIST\raw

['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
0/50 Test set: Average loss: 12.7659, Accuracy: 285/10000 (2.85%) lr=0.1
1/50 Test set: Average loss: 11.5047, Accuracy: 1135/10000 (11.35%) lr=0.1
2/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.1
3/50 Test set: Average loss: 11.5053, Accuracy: 1135/10000 (11.35%) lr=0.1
4/50 Test set: Average loss: 11.5054, Accuracy: 1135/10000 (11.35%) lr=0.080000000000
00002
5/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.080000000000
00002
6/50 Test set: Average loss: 11.5055, Accuracy: 1135/10000 (11.35%) lr=0.080000000000
00002
7/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.080000000000
00002
8/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.080000000000
00002
9/50 Test set: Average loss: 11.5053, Accuracy: 1135/10000 (11.35%) lr=0.064000000000
00002
10/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.06400000000
000002
11/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.06400000000
000002
12/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.06400000000
000002
13/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.06400000000
000002
14/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.05120000000
0000016
15/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.05120000000
0000016
16/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.05120000000
0000016
17/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.05120000000
0000016
18/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.05120000000
0000016
19/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.04096000000
000002
20/50 Test set: Average loss: 11.5053, Accuracy: 1135/10000 (11.35%) lr=0.04096000000
000002
21/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.04096000000
000002
22/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.04096000000
000002
23/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.04096000000
000002
24/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.03276800000
000001
25/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.03276800000
000001
26/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.03276800000
000001
27/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.03276800000
000001
28/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.03276800000
000001
```

```
29/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.02621440000
0000013
30/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.02621440000
0000013
31/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.02621440000
0000013
32/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.02621440000
0000013
33/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.02621440000
0000013
34/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.02097152000
000001
35/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.02097152000
000001
36/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.02097152000
000001
37/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.02097152000
000001
38/50 Test set: Average loss: 11.5052, Accuracy: 1135/10000 (11.35%) lr=0.02097152000
000001
39/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01677721600
0000008
40/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.01677721600
0000008
41/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01677721600
0000008
42/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01677721600
0000008
43/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01677721600
0000008
44/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01342177280
0000007
45/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01342177280
0000007
46/50 Test set: Average loss: 11.5050, Accuracy: 1135/10000 (11.35%) lr=0.01342177280
0000007
47/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01342177280
0000007
48/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01342177280
0000007
49/50 Test set: Average loss: 11.5051, Accuracy: 1135/10000 (11.35%) lr=0.01073741824
0000006
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
0/50 Test set: Average loss: 9.0398, Accuracy: 2960/10000 (29.60%) lr=0.1
1/50 Test set: Average loss: 7.6563, Accuracy: 3830/10000 (38.30%) lr=0.1
2/50 Test set: Average loss: 7.0765, Accuracy: 4454/10000 (44.54%) lr=0.1
3/50 Test set: Average loss: 6.5014, Accuracy: 5179/10000 (51.79%) lr=0.1
4/50 Test set: Average loss: 6.0995, Accuracy: 5882/10000 (58.82%) lr=0.0800000000000
0002
5/50 Test set: Average loss: 6.0026, Accuracy: 5957/10000 (59.57%) lr=0.0800000000000
0002
6/50 Test set: Average loss: 5.7164, Accuracy: 6292/10000 (62.92%) lr=0.0800000000000
0002
7/50 Test set: Average loss: 5.6437, Accuracy: 6295/10000 (62.95%) lr=0.0800000000000
0002
8/50 Test set: Average loss: 5.5421, Accuracy: 6348/10000 (63.48%) lr=0.0800000000000
0002
```

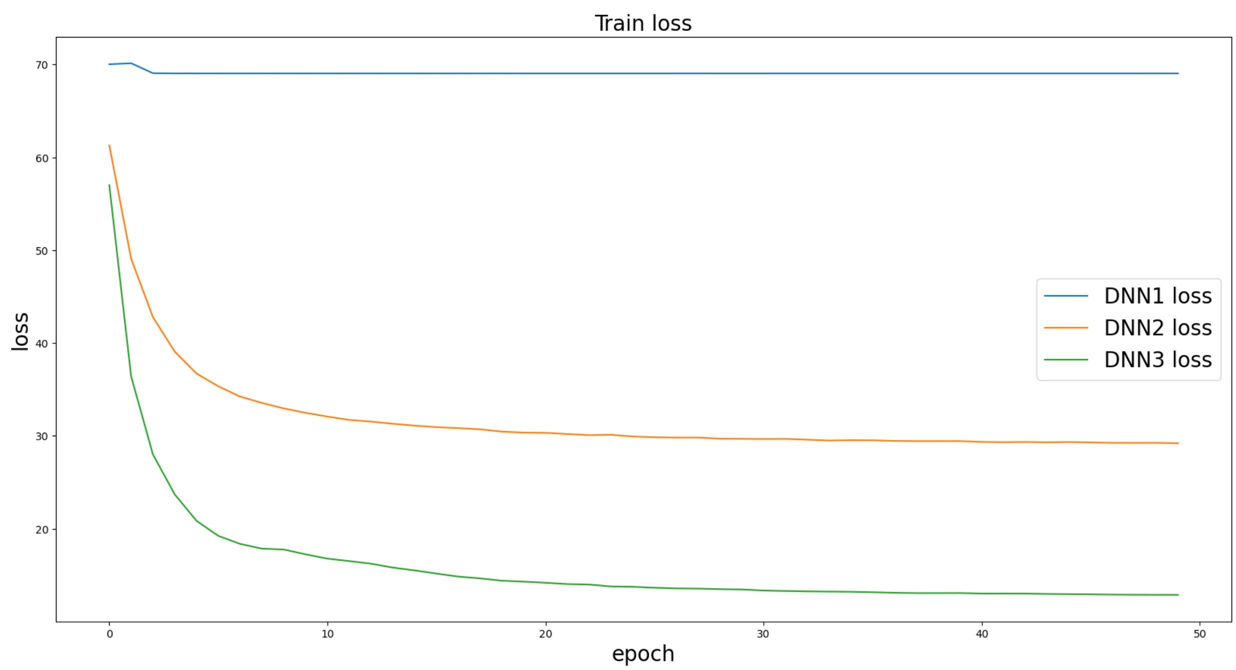
```
9/50 Test set: Average loss: 5.6111, Accuracy: 6225/10000 (62.25%) lr=0.0640000000000
0002
10/50 Test set: Average loss: 5.3333, Accuracy: 6596/10000 (65.96%) lr=0.064000000000
00002
11/50 Test set: Average loss: 5.3604, Accuracy: 6498/10000 (64.98%) lr=0.064000000000
00002
12/50 Test set: Average loss: 5.3549, Accuracy: 6539/10000 (65.39%) lr=0.064000000000
00002
13/50 Test set: Average loss: 5.3590, Accuracy: 6462/10000 (64.62%) lr=0.064000000000
00002
14/50 Test set: Average loss: 5.2215, Accuracy: 6663/10000 (66.63%) lr=0.051200000000
000016
15/50 Test set: Average loss: 5.2104, Accuracy: 6690/10000 (66.90%) lr=0.051200000000
000016
16/50 Test set: Average loss: 5.1478, Accuracy: 6726/10000 (67.26%) lr=0.051200000000
000016
17/50 Test set: Average loss: 5.1559, Accuracy: 6715/10000 (67.15%) lr=0.051200000000
000016
18/50 Test set: Average loss: 5.0924, Accuracy: 6806/10000 (68.06%) lr=0.051200000000
000016
19/50 Test set: Average loss: 5.0848, Accuracy: 6802/10000 (68.02%) lr=0.040960000000
00002
20/50 Test set: Average loss: 5.0581, Accuracy: 6830/10000 (68.30%) lr=0.040960000000
00002
21/50 Test set: Average loss: 5.0335, Accuracy: 6829/10000 (68.29%) lr=0.040960000000
00002
22/50 Test set: Average loss: 5.0758, Accuracy: 6819/10000 (68.19%) lr=0.040960000000
00002
23/50 Test set: Average loss: 5.0726, Accuracy: 6808/10000 (68.08%) lr=0.040960000000
00002
24/50 Test set: Average loss: 5.0139, Accuracy: 6852/10000 (68.52%) lr=0.032768000000
00001
25/50 Test set: Average loss: 4.9988, Accuracy: 6860/10000 (68.60%) lr=0.032768000000
00001
26/50 Test set: Average loss: 4.9991, Accuracy: 6831/10000 (68.31%) lr=0.032768000000
00001
27/50 Test set: Average loss: 4.9673, Accuracy: 6887/10000 (68.87%) lr=0.032768000000
00001
28/50 Test set: Average loss: 5.0061, Accuracy: 6830/10000 (68.30%) lr=0.032768000000
00001
29/50 Test set: Average loss: 4.9610, Accuracy: 6908/10000 (69.08%) lr=0.026214400000
000013
30/50 Test set: Average loss: 5.0023, Accuracy: 6846/10000 (68.46%) lr=0.026214400000
000013
31/50 Test set: Average loss: 4.9524, Accuracy: 6895/10000 (68.95%) lr=0.026214400000
000013
32/50 Test set: Average loss: 4.9554, Accuracy: 6916/10000 (69.16%) lr=0.026214400000
000013
33/50 Test set: Average loss: 4.9270, Accuracy: 6930/10000 (69.30%) lr=0.026214400000
000013
34/50 Test set: Average loss: 4.9609, Accuracy: 6878/10000 (68.78%) lr=0.020971520000
00001
35/50 Test set: Average loss: 4.9293, Accuracy: 6934/10000 (69.34%) lr=0.020971520000
00001
36/50 Test set: Average loss: 4.9359, Accuracy: 6934/10000 (69.34%) lr=0.020971520000
00001
37/50 Test set: Average loss: 4.9290, Accuracy: 6923/10000 (69.23%) lr=0.020971520000
00001
38/50 Test set: Average loss: 4.9569, Accuracy: 6893/10000 (68.93%) lr=0.020971520000
00001
```

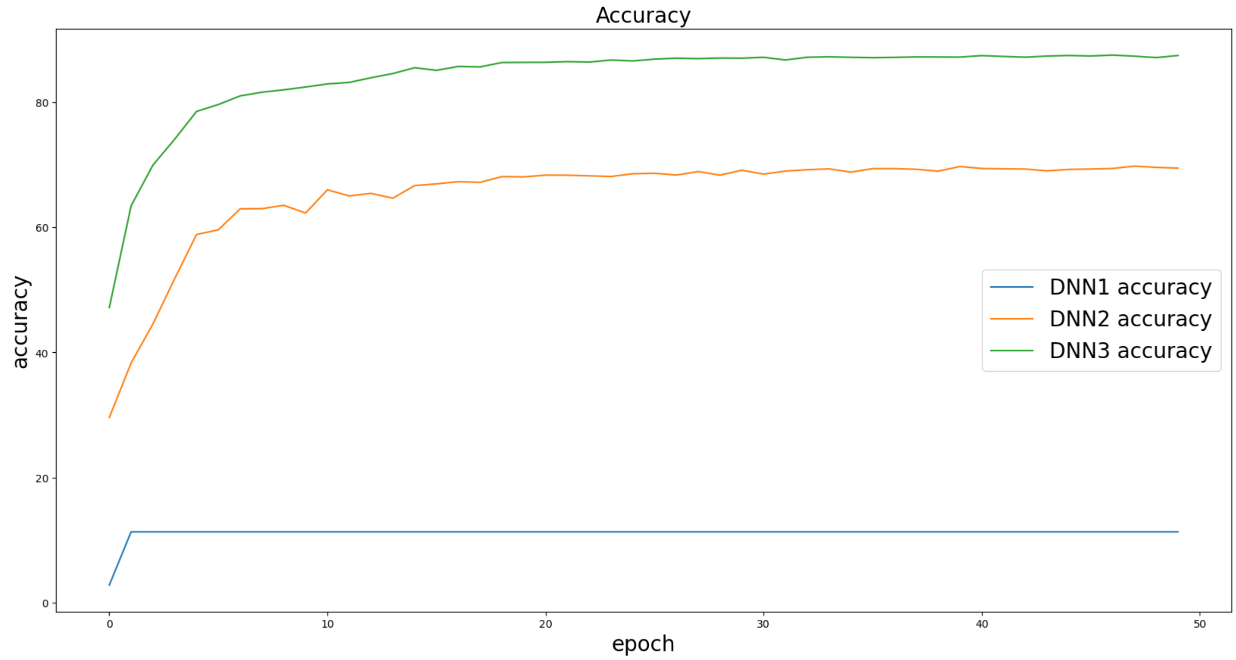
```
39/50 Test set: Average loss: 4.9226, Accuracy: 6968/10000 (69.68%) lr=0.016777216000
000008
40/50 Test set: Average loss: 4.9152, Accuracy: 6936/10000 (69.36%) lr=0.016777216000
000008
41/50 Test set: Average loss: 4.9140, Accuracy: 6931/10000 (69.31%) lr=0.016777216000
000008
42/50 Test set: Average loss: 4.9217, Accuracy: 6927/10000 (69.27%) lr=0.016777216000
000008
43/50 Test set: Average loss: 4.9236, Accuracy: 6899/10000 (68.99%) lr=0.016777216000
000008
44/50 Test set: Average loss: 4.9227, Accuracy: 6921/10000 (69.21%) lr=0.013421772800
000007
45/50 Test set: Average loss: 4.9282, Accuracy: 6928/10000 (69.28%) lr=0.013421772800
000007
46/50 Test set: Average loss: 4.9068, Accuracy: 6936/10000 (69.36%) lr=0.013421772800
000007
47/50 Test set: Average loss: 4.9065, Accuracy: 6974/10000 (69.74%) lr=0.013421772800
000007
48/50 Test set: Average loss: 4.9027, Accuracy: 6954/10000 (69.54%) lr=0.013421772800
000007
49/50 Test set: Average loss: 4.9003, Accuracy: 6942/10000 (69.42%) lr=0.010737418240
000006
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7
- seven', '8 - eight', '9 - nine']
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
0/50 Test set: Average loss: 7.3462, Accuracy: 4715/10000 (47.15%) lr=0.1
1/50 Test set: Average loss: 5.1601, Accuracy: 6340/10000 (63.40%) lr=0.1
2/50 Test set: Average loss: 4.3745, Accuracy: 6990/10000 (69.90%) lr=0.1
3/50 Test set: Average loss: 4.0608, Accuracy: 7405/10000 (74.05%) lr=0.1
4/50 Test set: Average loss: 3.4161, Accuracy: 7847/10000 (78.47%) lr=0.0800000000000
0002
5/50 Test set: Average loss: 3.2216, Accuracy: 7957/10000 (79.57%) lr=0.0800000000000
0002
6/50 Test set: Average loss: 3.0694, Accuracy: 8095/10000 (80.95%) lr=0.0800000000000
0002
7/50 Test set: Average loss: 2.9795, Accuracy: 8154/10000 (81.54%) lr=0.0800000000000
0002
8/50 Test set: Average loss: 3.0046, Accuracy: 8192/10000 (81.92%) lr=0.0800000000000
0002
9/50 Test set: Average loss: 2.9194, Accuracy: 8237/10000 (82.37%) lr=0.0640000000000
0002
10/50 Test set: Average loss: 2.8587, Accuracy: 8286/10000 (82.86%) lr=0.064000000000
00002
11/50 Test set: Average loss: 2.8997, Accuracy: 8310/10000 (83.10%) lr=0.064000000000
00002
12/50 Test set: Average loss: 2.7841, Accuracy: 8385/10000 (83.85%) lr=0.064000000000
00002
13/50 Test set: Average loss: 2.7167, Accuracy: 8452/10000 (84.52%) lr=0.064000000000
00002
14/50 Test set: Average loss: 2.6080, Accuracy: 8545/10000 (85.45%) lr=0.051200000000
000016
15/50 Test set: Average loss: 2.6326, Accuracy: 8503/10000 (85.03%) lr=0.051200000000
000016
16/50 Test set: Average loss: 2.5341, Accuracy: 8565/10000 (85.65%) lr=0.051200000000
000016
17/50 Test set: Average loss: 2.5236, Accuracy: 8558/10000 (85.58%) lr=0.051200000000
000016
18/50 Test set: Average loss: 2.4666, Accuracy: 8628/10000 (86.28%) lr=0.051200000000
000016
```

```
19/50 Test set: Average loss: 2.4607, Accuracy: 8630/10000 (86.30%) lr=0.040960000000
00002
20/50 Test set: Average loss: 2.4167, Accuracy: 8631/10000 (86.31%) lr=0.040960000000
00002
21/50 Test set: Average loss: 2.4156, Accuracy: 8642/10000 (86.42%) lr=0.040960000000
00002
22/50 Test set: Average loss: 2.3983, Accuracy: 8635/10000 (86.35%) lr=0.040960000000
00002
23/50 Test set: Average loss: 2.3580, Accuracy: 8668/10000 (86.68%) lr=0.040960000000
00002
24/50 Test set: Average loss: 2.3753, Accuracy: 8653/10000 (86.53%) lr=0.032768000000
00001
25/50 Test set: Average loss: 2.3503, Accuracy: 8682/10000 (86.82%) lr=0.032768000000
00001
26/50 Test set: Average loss: 2.3152, Accuracy: 8697/10000 (86.97%) lr=0.032768000000
00001
27/50 Test set: Average loss: 2.3173, Accuracy: 8690/10000 (86.90%) lr=0.032768000000
00001
28/50 Test set: Average loss: 2.3002, Accuracy: 8700/10000 (87.00%) lr=0.032768000000
00001
29/50 Test set: Average loss: 2.2926, Accuracy: 8697/10000 (86.97%) lr=0.026214400000
000013
30/50 Test set: Average loss: 2.2822, Accuracy: 8711/10000 (87.11%) lr=0.026214400000
000013
31/50 Test set: Average loss: 2.3060, Accuracy: 8669/10000 (86.69%) lr=0.026214400000
000013
32/50 Test set: Average loss: 2.2753, Accuracy: 8712/10000 (87.12%) lr=0.026214400000
000013
33/50 Test set: Average loss: 2.2700, Accuracy: 8720/10000 (87.20%) lr=0.026214400000
000013
34/50 Test set: Average loss: 2.2566, Accuracy: 8711/10000 (87.11%) lr=0.020971520000
00001
35/50 Test set: Average loss: 2.2720, Accuracy: 8705/10000 (87.05%) lr=0.020971520000
00001
36/50 Test set: Average loss: 2.2522, Accuracy: 8710/10000 (87.10%) lr=0.020971520000
00001
37/50 Test set: Average loss: 2.2676, Accuracy: 8718/10000 (87.18%) lr=0.020971520000
00001
38/50 Test set: Average loss: 2.2566, Accuracy: 8717/10000 (87.17%) lr=0.020971520000
00001
39/50 Test set: Average loss: 2.2499, Accuracy: 8715/10000 (87.15%) lr=0.016777216000
000008
40/50 Test set: Average loss: 2.2434, Accuracy: 8738/10000 (87.38%) lr=0.016777216000
000008
41/50 Test set: Average loss: 2.2351, Accuracy: 8725/10000 (87.25%) lr=0.016777216000
000008
42/50 Test set: Average loss: 2.2420, Accuracy: 8714/10000 (87.14%) lr=0.016777216000
000008
43/50 Test set: Average loss: 2.2360, Accuracy: 8731/10000 (87.31%) lr=0.016777216000
000008
44/50 Test set: Average loss: 2.2247, Accuracy: 8740/10000 (87.40%) lr=0.013421772800
000007
45/50 Test set: Average loss: 2.2311, Accuracy: 8732/10000 (87.32%) lr=0.013421772800
000007
46/50 Test set: Average loss: 2.2158, Accuracy: 8747/10000 (87.47%) lr=0.013421772800
000007
47/50 Test set: Average loss: 2.2333, Accuracy: 8729/10000 (87.29%) lr=0.013421772800
000007
48/50 Test set: Average loss: 2.2497, Accuracy: 8707/10000 (87.07%) lr=0.013421772800
000007
```

```
49/50 Test set: Average loss: 2.2227, Accuracy: 8740/10000 (87.40%) lr=0.010737418240
000006
```

In [24]:
```
# Plot loss & acc
plt.figure(figsize=(20,10))
plt.plot(trainloss_MNIST1, label='DNN1 loss')
plt.plot(trainloss_MNIST2, label='DNN2 loss')
plt.plot(trainloss_MNIST3, label='DNN3 loss')
plt.xlabel('epoch',fontsize=20)
plt.ylabel('loss',fontsize=20)
plt.title('Train loss',fontsize=20)
plt.legend(fontsize=20)
plt.show()

plt.figure(figsize=(20,10))
plt.plot(accuracy_MNIST1, label='DNN1 accuracy')
plt.plot(accuracy_MNIST2, label='DNN2 accuracy')
plt.plot(accuracy_MNIST3, label='DNN3 accuracy')
plt.xlabel('epoch',fontsize=20)
plt.ylabel('accuracy',fontsize=20)
plt.title('Accuracy',fontsize=20)
plt.legend(fontsize=20)
plt.show()
```




Accuracy
accuracy
epoch
DNN1 accuracy
DNN2 accuracy
DNN3 accuracy
0
20
40
60
80
0
10
20
30
40
50

In [ ]:

In [ ]: